什么是 *Getting Real?*

想构建一个成功的 Web 应用么? 那么正是时候 Getting Real. Getting Real 是一种更小规模，更快速，更高质量的软件构建方法。

- Getting Real 是关于省略所有表达现实（图表，曲线，矩形，箭头，统计图），而构建现实。
- Getting real 是追求精炼。更少的代码量，更少的软件，更少的功能，更少的文档工作，更少无所谓的东西（而且大部分你认为必要的，其实不是）。
- Getting Real 是保持精益，变得敏捷。
- Getting Real 从界面开始，也就是用户使用的屏幕。它从实际的用户体验开始，并且构建似曾相识的体验。这让你在软件误入歧途之前得到正确的用户界面。
- Getting Real 是关于迭代和降低变化成本的方法。Getting Real 基本上是关于上线，调整，持续改进，其目标的开发 Web 软件的最佳途径。
- Getting Real 只交付客户所需的，摒弃任何客户不需要的。

Getting Real 的优点

Getting Real 能够交付更好的结果，是因为它强迫你处理真正要解决的问题，而不是关于那些问题的空想。它迫使你面对当下。

Getting Real 更注重实际的用户界面，而不是功能规格说明书和其他昙花一现的文档。只有当一个真实的网页呈现出来，相关的功能规格才是可信的，被证明是可接受 的。那才是是我们的客户将要看到和使用的。那才是需要关心的。Getting Real 帮助你更快达到这个目的。并且那意味着你正在基于真实需求，而不是异想天开来构建软件。

最后，Getting Real 是适合于 Web 软件的理想途径。那种把软件包装在盒子里，再等一年到两年才发布一个更新的学院派方法已经过时了。不像需要安装的软件，Web 应用能够以天为单位持续改进。Getting Real 利用了这种优势来提升 Web 应用的价值。

如何编写健壮的软件 How To Write Vigorous Software

健壮的著作是简明的。句子无废词，段落无废句子。同样的原因，画应无多余的线条，机器应无多余的零件。这不是要作者刻意缩句来逃避细节，从而提纲挈领，而是要作者字字珠玑。

—来自 "The Elements of Style" by William Strunk Jr.

不再发胖

旧方式：冗长，官僚主义的，“我们正在这么做来控制这些蠢驴” 的流程。典型后果是：臃肿，过目即忘，平庸得掉渣的软件。Blech。

Getting Real 除掉...

- 花费数月，甚至数年的进度表
- 不切实际的功能规格文档
- 可伸缩性的争论

- 又臭又长的员工大会
- 大量招人的需求
- 毫无意义的版本号
- 憧憬完美未来的幼稚“路线图”
- 无穷尽的偏好设置选项
- 外包支持
- 不切实际的用户测试
- 写无用文档
- 自顶向下的管理结构

你不需要成吨的钞票或者庞大的团队或者漫长的开发周期来构建伟大的软件。那些正是缓慢，晦涩，变化成本高昂的应用程序的帮凶。Getting real 反其道而行之。

这本书将带给你...

- 信仰之重要
- 为什么小是好事情
- 怎样构建更少
- 怎样从现实世界中快速找到创意
- 怎样培养团队
- 为何要由内到外的设计
- 为什么写作至关重要
- 为什么要比对手少做
- 如何升级你的应用和散播文字
- 成功维护的秘诀
- 发布后能够持续保持后劲的秘诀。
- 其他...

本书关注于理论高度。我们不会使你陷入代码片段细节，或者是 CSS 窍门。我们会坚守在驱动 Getting Real 过程的主要思想和价值观上。

本书适合你么？

你是管理者，设计师，程序员，或者市场人员。

你意识到旧规则不再管用了。每年通过光盘分发你的软件？2002 这个版本号怎么样？

或者你对敏捷开发和企业组织结构略懂皮毛，但是热切的想多了解一些。

如果听起来你是其中之一，那么这本书就是为你准备的。

注意：虽然这本书着重在构建 Web 应用上，很多理念也可以应用到非软件活动。关于小型团队，快速原型，期望迭代，和许多提到的其他经验能够为你引 路。无论你正在开始一项业务，写一本书，设计一个网站,记录签名册，还是其他各种各样的活动。一旦你在你生活中的某一领域开始 Getting Real，你就或发现这些概念能适用的非常广泛。

关于 37signals

我们做什么

37signals 是一个创造简单的，专一的软件的小团队。我们的产品帮助你协同工作，组织团队。超过 35000 个人和企业使用我们的 Web 应用来搞定他们的业务。 来自华尔街杂志的 Jeremy Wagstaff 写到 “37signals 的产品是超简单，精致，直接了当的工具，这些工具让微软的 Outlook 软件使用起来像受刑 。”我们的软件不会把你推到这种境地。

我们的习惯做法

我们相信软件太复杂了。太多的功能，太多的按钮，需要学习太多东西。 我们的产品比对手做的少 — 故意地. 我们构建的产品运行灵巧，感觉舒适，允许你以自己的方式做事，并且容易使用。

我们的产品

当这本书出版之即，我们有 5 个商业产品和一个开源框架。

Basecamp 把项目管理作为首要问题。Basecamp 提供了消息板，待办事宜，简单调度，协同写作，文件共享。 而不是甘特图，炫丽的曲线图，和繁重的电子表格。目前，成千上万的人同意这是一种更好的方式。来自 Salon.com 的 Farhad Manjoo 说：“Basecamp 代表了 Web 软件的未来。”

Campfire 提供了业务模式下的简单群聊方式。实时持久的群聊对于业务来说非常重要。传统的实时聊天对于快速的一对一模式很有效。但是对于 3 个或者更多的人同时聊天来说异常痛苦。Campfire 解决了此问题和其他相关问题。

Backpack 是一种替代那些玄乎，复杂，“通过 25 个步骤管理人生”之类的个人信息管理系统的产品。Backpack 在页面，笔记，待办事宜，电话和电子邮件通知上的 简单尝试，在受“statis-quo-itie”折磨的一类产品中，是一个独具匠心的创意。Wall Street Journal 的 Thomas Weber 说它是同类产品中最出众的。 New York Times 的 David Pogue 说它是一个“非常酷”的组织工具。

Writeboard 使你能够撰写，分享，修订，和比较自己或者他人的文章。臃肿的文本处理工具，对于你 95%的文字是功能过剩的，而 Writeboard 是一个全新的替代品。Web-guru Jeffrey Zeldman 说：“37signals 的天才思想王者归来。”

Ta-da List 维护聚合你的所有待办清单，并且以在线方式组织。为你自己维护待办清单，或者通过和其他人分享来协作。没有更好的方式来搞定这些了。迄今为止，其创建了超过 100，000 个清单和 1，000，000 项行动。

Ruby on Rails, 对于开发者来说，是一个用 Ruby 编写的全栈式的开源 Web 框架。其使得开发真是应用快速而简单。你可以关注在你的思想上面，而由 Rails 操心杂事。 O’Reilly 的 Nathan Torkington 说：“Ruby on Rails 太令人震撼了。使用它像是观赏一个功夫片，片中一堆流氓框架准备痛扁这个小新人，没想到却被各种充满想象力的方式揪住了屁股。”Gotta 喜欢 这段话。

你可以从 www.37signals.com 找到更多关于我们产品和我们的公司的信息.

告诫，免责，和其他丑话说在前头

为了扫清障碍，下面是我们对于一些不时听到的抱怨的答复。:

"这些技术不适合我"

Getting real 是一套对我们来说效果非凡的系统。但是本书的思想并不是放之四海皆准。如果你正在构建一套武器系统，一个导弹控制设备，一个为数以百万用户服务的银行系 统，或者其他对生命、财产至关重要的系统，你将会回避一些我们的放纵主义态度。请继续前行并且采取一些其他的防范措施。

而且不必全部采纳或者全盘否定我们的主张。即使你没能力完全 Getting Real，一定有少许观点你能够偷偷摸摸避开当权者而实行。

"这些思想不是你们发明的"

我们没有声明我们发明了这些技术。许多概念已经以各种形式伴随我们很久了。当你读到一些我们的建议，并且它提醒你读到 的一些东西已经在一些人的日记或者一些已经出版了 20 年的书 1cc;面了。这是完全可能的。这些技术并不是 37signals 的独创。我们只是告诉你我们怎样工作和是什么带给我们成功的。

"你们的观点过于绝对"

如果我们的口吻看起来好像无所不知，目中无人，请宽容我们。我们认为果敢地提出观点要比唯唯诺诺，模棱两可要好得多。 如果这是骄傲自大的形象，它就是。我们宁愿具有煽动性，也不愿意用“那要看…”这样的话来和稀泥。当然这些规则需要时间来完善或者打破。而且一些策略可能 不适合你的场合。请运用你的判断力和想象力。

"我们公司内部不适用."

觉得你的公司太大以至于难以 Get Real？连微软也 Getting Real（而且我怀疑你的公司更大）。

即使你的公司典型地执行长期，大型团队的调度计划，仍有方式 Get real。第一步是分解成更小的团队。当太多的人牵扯进来，什么事都搞不定。你越轻装上阵，事情就做的越快越好。

没错，这需要一些推销潜质。让你们的公司投身于 Getting Real 过程中。给他们出示这本书。向他们炫耀你用更少时间和更小团队所得到的真实成就。

解释 Getting Real 是一种尝试新概念的低风险，低投入的方式。

如果你有勇气的话，秘密行动。在雷达下面飞行来证明真实结果。这正是 Start.com 团队在微软运用 Getting Real 的方式。“我观察过 Start.com 团队的工作方式，而他们没有经过许可”，微软的技术传播者 Robert Scoble 说。“他们有一个老板作为空中掩护。他们每次只接受一点功能，实现他们，并且响应反馈。”

推进微软的 Start.com

在大公司中，流程和会议是非常平常的。数月的时间被浪费在规划功能，争论细节，力求达到每个人在什么是对于客户来说是“正确”的东西上达成一致。

这可能是塑料包装软件的正确途径，但是基于 Web 的软件有难以置信的优势。立刻发布！让用户告诉你这是不是正确的东西。如果你愿意，你可以当天修正它然后发布最新版。没有话比客户的意见更有用 — 拒绝举行罗嗦的会议和争论。仅仅发布产品，并且证明这个观点。

知易行难 — 这说明：

数月的规划没有必要。
花费数月来写规格说明根本没必要 — 规格说明应该在开发过程中理清框架，描述并且精化细节。不要试图在开发开始之前解决所有选而悬而未决的问题，并且敲定所有细节。

发布更少，但高质量的功能.
你不需要一道电闪雷鸣伴随着全新的发布和一捆新功能。一小块一小块地喂用户，让他们能够消化。

如果发现了细微的 bug，先发布敲定的核心功能，然后发布补丁。动作越快用户反馈越好。纸上谈兵听起来不错，但是实践中往往不理想。你越快发现观点的关键错误越好。

一旦你快速迭代，并且响应用户反馈，你就和用户建立了一种关系。记住目标是通过构建用户所想来赢得客户。

—*Sanaz Ahari, Start.com, Microsoft 的程序经理*

建构从简

做得比竞争对手少

常规的思维方式告诉我们，不管竞争对手做什么你总是要比他们加多一些。如果他们有 4 个特色功能，你就需要做出 5 个（或 15 个，或 25 个）。如果他们花了 x，你就该花 xx。如果他们有 20，你就得有 30。

这种强调更多一层的冷战竞争思维是行不通的死胡同。如此创造产品的方式是昂贵的，过分防御的，并且有点偏执不正常的。防御性的偏执的公司是做不到前瞻性思维的，他们只能做事后思维。他们不可能领导，只能跟从。

如果你只想创建一个随大流的公司，那么你现在就可以放下这本书，无需继续读下去了。

那怎样才是有效的方法呢？答案是：做少。靠做得比对方少来打败他。解决简单的问题，把繁复困难棘手的问题留给大众。不做更多，相反的我们做的更少。不赶超，相反的我们试着退一步，守后。

我们将会将贯穿全书论述“少做”的概念，现在先简要介绍“少做”的含义作为热身：

- 更少的功能
- 更少的选择项和首选项
- 更少的配备人员和企业架构
- 更少的会议和抽象讨论
- 更少的承诺

是什么问题一直困扰你？

为自己而做这个软件

一个很好的做软件的方式就是一开始用它来解决你自己的问题。由于你自己变成了软件的目标受众因此你会知道什么是重要的什么不是。这样做下去将会是推出一个突破性产品的伟大起始点。

关键是你要了解你并不孤单。如果你有一个困扰你的问题那么非常有可能成百上千的其他人业有同样的烦恼。这，就是你产品的市场。看，不难找到吧？

Basecamp 这个产品来自于一个困扰我们的难题：做为一个设计公司，我们需要有一个很简单的方式来和客户做项目沟通。一开始，我们建了一个外部 网，通过不断更新其内容来连线客户。但每次一个项目需要更新我们就得手动更改 HTML，这实在不是一个解决方案。这些项目网站总是看起来不错但最终总是被 放弃了。这是很令人恼火的，因为它使我们变得很没有组织性，也让客户感到无所适从。

于是我们开始寻找其他解决方案。但每样我们找到的工具都是 1)并不能做我们想做的 或 2)充斥着我们所不需要的功能 — 比如象账单，登录权限控件，图表，等等。我们觉得一定会有更好的方案，最终我们选择了自己来做这个软件

当你在做软件解决你自己问题的时候，你创造的工具是你对之有激情的。那激情就是关键所在。激情意味着你真正去用这个软件，去关心这个软件。这是能感动他人并一起为之所动的最好的方式。

自助（自挠其背而止痒）

这是很长一段时间以来被开源领域奉为箴言的 — 他们叫它做“自挠其背而止痒”。对开源软件开发者来说，这意味着他们将自己去组织必需的工具，用自己的方式来推出产品。不仅于此，这样做的有着更深远的裨益。

作为一个新软件的网页设计师和程序开发者，每天你要面对上百个细小的决策：是要蓝色的还是绿色的？用一个表格还是两个？静态的或是动态的页面？放弃 重新来过还是修复？一般我们是怎样来做这些决定的呢？如果那是我们认为很重要的我们也许会提出来讨论。其余的我们就靠猜想。最终所有那些猜想的部分将积累 成软件的一种债务 — 一堆互相交织着的错综复杂的充满不确定因素的网。

作为一个开发者，我厌恶这种现象。想到有那么多的小型定时炸弹分布在这个软件中，给我带来了很大的压力。 自己帮助自己的开源软件开发者不会有这样的困扰。因为他们就是用户本身，90%的情况下他们了解他们应该做什么决定。我想这是为什么这些人能够在一天辛苦 的工作回家后还能继续编写开源程序的众多原因之一，因为它反而是一种放松。

—Dave Thomas, 一个实用主义编程者

一切源于必要性

Campaign Monitor 公司的成立事实上是来源于必要性。多年来各种 email 市场营销工具的低劣品质一直困扰着我们。一个工具可以做到 x 和 y 却总也做不到 z，另一个能搞定 y 和 z 却总也做不好 x。老是这么下去我们将无法成就赢家。

我们于是决定整顿我们的时间表，着手开发我们理想中的 email 市场营销工具。我们很清醒地决定不看其他人在做什么而是专心做一个能让我们自己和客户都活得自在一些的产品出来。

事实证明，我们并不是惟一对现状感到不满的人。我们对软件成品做了一些改动，这么一来所有的设计公司（不仅我们自己）都能使用它并且开始帮我们推广。不到 6 个月，数以千计的设计师开始用 Campaign Monitor 来发布他们自己和客户的 email 推广信了。

—David Greiner, 创始人, Campaign Monitor

你必须在乎这个东西

当你在写一本书，你必须要有一个以上的有趣故事可讲。你必须有强烈地要讲叙这个故事的欲望。你必须要以某种方式把自己投入到故事中去。如果你要和一件事业共存两年，三年或你地一生，你必须要真正在乎它

—Malcolm Gladwell, 作者 (from 杂看 Malcolm Gladwell)

找自己募资

外部资金只是第二条路(plan B)

很多创业者的首要任务就是找投资者募集资金。但必须记住，当你寻求外部资金的时候就意味着你不得不向别人汇报。于是别人对你的期望值将提升。投资者无不希望他们能够收回资金 — 并且是以最快的速度收回。导致一个不幸的事实就是往往抢钱优先过做一个优质的产品。

现时创业并不需要太多的启动资金。硬件便宜了，大量的优秀框架软件也都开源免费了。而且创业的激情是不需要标价的。

因此手头有多少钱就先动起来做多少事。用心想想决定什么是你最基本需要的，什么是可以先舍弃不做的。什么事是你可以三个人就搞定而不必用到十个人？什么是两万块而不用十万块就能办到的？什么样的软件你可以一边白天打工用业余时间做出来的？

资源拮据往往能激发想象力

当在有限的资源平台上运作，你往往会被迫更早的更紧迫的认识到这种压力。那是一件好事。有压力才会促使创新。

拮据也能迫你更早地将你的点子推出来 —这是另一件好事。这么跨出去一两个月后你就会比较清楚的了解这个点子是否有戏。这样一来，你就会在短时间内树立起自信心，不再那么急切寻求外部资金了。 如果你的点子行不通，是虚的（酸柠檬），那么就是重新来过的时候了。坏消息至少你现在知道比几个月后或几年后知道好。至少你比较容易退出。当有投资者涉入 的时候退出方案要复杂的多了。

如果你做软件只是想搞一度快钱，那么事实是瞒不住的。想很快的回本实践中是不大可能的。所以，专心做一个优质的软件，做出一个你和你的客户都能长久依赖的工具才是正道。

两条路

[Jake Walker 拥有的一家公司是用投资者的钱创立的 (Disclive) 另有一家不是 (The Show)。在这里他探讨了这两条路的不同风景。]

所有问题的根本并不是筹募资金本身，而是随之而来的一系列事情。基本上就是更高的期望值。人们于是开始领工资，然后动力就成了做一个软件然后把它卖 了，或想办法把种子基金给还了。就我前面的第一家公司（Disclive)，出于必要性我们起步得比我们原先设想的高很多 —

[关于第二家公司 The Show] 我们发现我们可以推出一个花更少的钱却可以做得更好得产品，只不过要多花些时间。我们把很多自己的钱赌到一个人们愿意牺牲一点时间来换取品质的产品上面。 但公司一直保持着小规模（也预计会一直这么走下去）。从那以后，我们就全部是内部募资。通过采取一些灵活的交易方式和我们的供应商合作，我们从不需要投入 自己很多的资金。并且我们并不是做大后卖了，而是随需要而发展，走持续的可盈利道路。

—评论来自于 Signal vs. Noise

固定时间和预算，但灵活控制产品外延

准时地在预算内推出

一个简单方法让你能准时地在预算范围内推出产品：定额定量。绝对不要在一个难题上多投时间和金钱。要么缩小规模，要么缩小范围。

总会有一个这样误区：我们能做到准时地，在预算内发布一个规模完整的产品。这可以说完全不可能的，如果真是这样的话一定是以质量做为代价的。

如果你不能在预定的时间和预算内完成所有的东西的话，就不要拉长时间和增加预算。相反的，把产品的外延缩小些。下一步总是有时间可以加东西进去 — 过后有的是时间，当下却是稍纵即逝的。

做一个比预计要小巧些的好东西比做一个庞大平庸而又漏洞百出的东西要现实的多，因为你要象魔术师一样巧妙的照顾到时间，预算和产品内容的方方面面。变魔术就交给 Houdini（魔术大师）。你所做的可是在运作一个真正的事业，在推出一个真实的产品。

以下是一些固定时间和预算，灵活控制产品外延的好处:

- **要有优先级**

  你一定要搞明白什么才是最重要的。什么是首发的产品中必须要具备的特性功能？这个限制思维逼迫你下一些痛苦但必要的决定，而不是挑挑拣拣的拿不定主意。

- **要现实些**

  设定期望值是关键。如果你同时要固定死时间，预算和产品的外延，你将不能推出高层次的产品。当然你可能推出个东西，但那个“东西”会是你真正想做的吗？

- **要有灵活性**

  及时应变的能力很重要。如果什么都固定死就很难应变。给产品的外延注入机动性，当真正做起来就有比较多的选择空间。机动灵活是你的良师益友。

我们建议：范围缩小些。做半个产品比做半拉子的产品好（后面将进一步论述这点）。

一天，两天，三天...

知道一个项目是怎样拖到最后比预计迟一整年的吗？就是这么一天一天拖出来的。

—Fred Brooks, 软件工程师，电脑科学者

找个敌人

挑选一场战斗

有时了解你的应用程序应该做成什么样子的最佳方式就是，认识到它不应该成为什么。搞清

楚你的软件的对手是谁，就象点一盏灯，能照亮你前行的道路。

当我们决定开发项目管理软件的时候，我们清楚的知道微软项目软件（Microsoft Project)是这个小舞台上的一个庞然大物，大猩猩。我们不是去害怕这个大家伙，相反的我们把它做为一个刺激我们前进的引擎。我们决定 Basecamp 将做成一个完全不同的项目管理软件，就是反 Microsoft Project 而行。

我们意识到项目管理并非就是有关图表，报告和统计数据 — 它更重要的是沟通。它并不是要一个项目经理坐得高高在上去传达一个项目计划。它应该是每个人都参与的，一起担起责任来使项目付诸实现。

我们的敌人就是项目的独裁者和他们用来指手画脚的工具。我们要把项目管理民主化 — 让每个人都能成为它的一份子（包括客户）。只有每个人都承担负责流程的一部分，这样整合起来项目才能做得更好。

说说 Writeboard 这个协同文字编辑软件，我们知道有很多的竞争对手的产品内建了许多复杂玄妙的功能。正因为如此，我们决定着重从“不花哨” 入手。我们开发了这个程序仅仅来让人们共享和协作，而不用一些非必要的功能来使他们举步维艰。只要是不必要的我们就不做。推出后仅 3 个月的时间，已经超过 10 万个用户在使用 Writeboards。

当我们开始着手 Backpack 这个资讯管理软件的时候，我们的敌人就是严格的组织框架和规章制度。人们应该要能够用他们自己的方式来管理他们的信息 — 而不是在一堆预先规定好的格式和众多的表格中填空。

你能从敌人那里得到的一个好处就是：一个非常清晰的营销理念。人们很容易被冲突对立挑动。并且通过把一个产品和另一个作比较能更多地了解这个产品。 选中了这么一个敌人，你给人们灌输了他们想要知道的对立的信息。这样一来，他们不仅能更好更快地认识你的产品，也会站到你的这边。这是一个吸引注意力和引 发产品倾向性的一个万无一失的方法。

说了这么多，必须指出的是，也不要太过着迷于竞争。过分地去分析其他产品会慢慢限制你的思维想像力。很快地看一下他们在做什么，然后就要回到你自己地理念和理想上来。

别老跟着领头羊

营销人员 (和几乎所有人) 都被培训要跟从领导者。自然的本能都是在思考竞争对手做对了什么，然后你在那个基础上做得更超过。 — 如果你的对手在竞价你就一定要比他更便宜，如果他在竞速你就要比他做得更快。这么一来出现的问题是万一消费者听信了别人的故事（或谎言），你再要把他说转 回来就会象要说服他承认他是错的一样。没人喜欢承认他是错的。

于是你应该创造一个不同的故事来说服听众，告诉他们你的故事要比他们在其他地方听到的更重要。如果你的竞争对手是在竞速，那么你就必须转到竞价上 来。如果他们强调的是健康，那么你就必须推销方便性。并不是简单地在 x/y 轴图表上标出“我们比较便宜”的声明，而是实实在在地用真实的完全不同于他人正 在推销的故事。

—*Seth Godin, 作家/企业家(from 做个更好的骗子)*

问题的关键是什么？

老是看你的竞争对手在做什么是你给自己找麻烦的最快的方式之一。对于我们建造 BlinkList 这个平台来说尤其确实。从我们推出以来已经有其他 10 个类似的社交关系网软件相继出现。有些人已经开始在网上用并排图表来做不同软件之间功能特色的详细比较。

这是很容易误导的。我们不随大流，相反的，我们只看大方向，时时提醒自己什么是我们想要解决的问题关键，怎样去解决它。

—Michael Reining, 共同创立人, MindValley 和 Blinklist

它不该成为一种交易

你的激情 — 或者冷漠 — 会起作用的

如果你越不把软件当作一种交易去做，你就越能做得好。把它控制在一个你能把握的小范围内，你就有可能真正地享受过程。

如果你做这个软件一点不兴奋，那就是什么地方出了问题了。如果你只为了赶紧抢点钱做掉它，那这种影响会出现在最终产品上。同样地，如果你满怀热情地去做它，结果也会反映在产品上。人们是能够分辨出个中的区别的。

激情的体现

在设计这个高度主观，具争议性且难以界定的领域里，没有什么是能做到比表达激情更直接清晰的了。一个产品会使你欢欣或让你无动于衷是很显而易见的; 不管是前者或后者，要人们不发现软件背后那些创造者的感情投入是很困难的。

热情是很容易凸显出来的，漠然也是同样难以掩饰的。如果你并非带着一种诚挚的心去对待手头上的产品，那么它留下的漠然与空白是几乎不可能掩饰的，不管一个设计作品表面看来多么精细吸引人。

—Khoi Vinh, Subtraction.com

烘培师

在美国，在这个时代生意经往往是提出一个构想，让它能盈利，在还能盈利的时候把它卖了，然后改行或不干了。这往往是一个能否挺下去的问题。对我而 言: 去热爱烘培，卖你自己做的面包，人们如果喜欢它我就卖多些。我就这么把烘培事业做下去，因为我知道我在做好吃的人们爱吃的东西。

—Ian MacKaye, Fugazi 的会员， Dischord Records 的合伙人
(from Salon.com People | Ian MacKaye)

更小的质量

你越做到精益，改变越容易

一个物体的质量越大，改变方向需要的能量越多。物理世界的这个真理同样适用于商业世界。

当真理应用到 Web 技术的时候，改变必须是简易和低廉的。如果你不能如飞一样的改变，你就会败给能够做到的竞争者。这就是你需要追求更小质量的原因。

质量会由于以下因素增加

- 长期合同
- 多余的职员
- 固执的决策
- 关于会议的会议
- 厚重的流程
- 存货（物理的或者头脑的）
- 硬件，软件和技术的锁定
- 专有数据格式
- 未来被过去支配
- 长期的路线图
- 办公室政治

质量会由于以下因素减少

- 必要而及时的思考
- 多面手的团队成员
- 拥抱限制，而不是试着移除他们
- 更少的软件，更少的代码
- 更少的特征
- 小规模团队
- 简单
- 被拆分为正交的接口
- 开源产品
- 开放的文件格式
- 开放的文化，使承认错误更容易

更小的质量使你快速的改变方向。你可以随机应变和演进。你可以集中于好的主意，摈弃坏的。你可以倾听并尊重你的客户。你可以集成新技术现在而不是以后。你驾驶的是蒸汽船而不是飞机货舱。为这个事实骄傲吧。

举例来说，想象一个精益，小质量的公司，用更少的软件，更少的特征制造了一个产品。另一方面是一个更大质量的公司拥有一个带有明显更多软件和特征的 产品。那么试想一下，随着新技术，比如 Ajax，或者新概念，比如标签的出现，谁会更快的调整他的产品？拥有更多软件更多特征还带着 12 个月路线图的团 队，还是另一个团队，拥有更少软件更少特征和一个更有机的流程——“让我们集中于我们当下应该集中精力的地方吧”。

很明显，更小质量的公司在适应市场真实需要的方面处于更有利的位置。当小质量公司已经完成转换很久以后，更大质量的公司有可能仍然在争论变化或者将他们推入官僚主义的流程中。小质量公司会领先两步于仍然在争论如何走的大质量公司。

反应快，敏捷，小质量的公司可以快速的改变他们整个业务模型，产品，特征集和营销信息。他们可以犯错并快速的修复。他们可以改变他们的优先级，产品组合和重点。还有最重要的，**他们可以改变他们的想法** 。

减少改变的成本

通过减少改变的阻碍保持灵活

改变是你最好的朋友，改变的代价越大，你越不可能做出改变。如果你的竞争对手可以比你更快的改变，你会处于一个很大的劣势。如果改变变得过于昂贵，你已经死了。

这就是保持精益真正帮助你的地方。很短时间内改变的能力是小团队与生俱来而大团队永远不会有的。这是大家伙嫉妒小家伙的地方。让巨大组织里的大团队花费数周才能改变的，对于小团队可能只需要 1 天。这个优势是无价的。低廉和迅速的改变是小团队的秘密武器。

请记住：所有的现金，所有的营销策略，所有的世界上的优秀人物，都买不到小带来的敏捷。

顺其自然

顺其自然是敏捷的根本原则之一，也是最接近纯正魔术的一个。顺其自然的特性不是设计的或内建的，它自然的发生，就如系统其余部分的动态结果。“顺其 自然”来自于 17 世纪中期的拉丁文，意思是“无法预见的出现”。你不可能为它做计划，或者做时间表，但是你可以为它营造一个环境，让它发生并受益于它。

顺其自然一个经典的例子是鸟群的迁徙行为。计算机模拟只需要少至三个原则（按照“不要撞到一起”），你会一下子得到非常复杂的行为来描述鸟群在天空 中优雅的飞行和滑翔，重组队形绕开障碍，等等。没有一个高级行为（比如躲开障碍重组形成的相同形状）是被规则规定的；都是由系统动态衍生的。

简单的规则，就像鸟群的模拟一样，导致复杂的行为。复杂的规则，就像许多国家的税法一样，导致愚蠢的行为。

许多常见的软件开发实践带来了令人遗憾的边际效应，他们消除了顺其自然行为的发生机会。大多数优化的尝试——直率的敲下一些代码——减少了互动和关 联的宽度和范围，而这些正是顺其自然的来源。在鸟群迁徙的例子中，正如一个设计良好的系统，正是互动和关联创造了有趣的行为。

我们把事物绑的越紧，留给创新的、顺其自然的解决方式的空间就越少。不管是在需求被完全理解前就锁定了需求，还是不成熟地优化代码，让终端用户使用系统前引进了复杂的导航和工作流场景，结果都是一样的，一个过分复杂、愚蠢的系统，而不是顺其自然带来的清洁和优雅的系统。

保持小，保持简单，顺其自然。

—安德鲁·亨特(Andrew Hunt)， *实效程序员网站(The Pragmatic Programmers)*

三个火枪手

用三人小组构建 1.0 版本

对于产品的 1.0 版本，请从只有三个人开始。三是一个魔力数字，提供足够人力的同时允许你保持流畅和敏捷。从一个开发者，一个设计者，和一个清道夫（一个可以在开发和设计中随意切换的人）开始。

现在显而易见的是用很少的人建造一项应用是一个挑战。但是如果你有一个正确的团队，挑战是值得的。有才能的人不会需要无尽的资源。他们会在约束限制 下的工作和利用创造力解决问题的挑战中成功。缺少人力意味着你会被迫更早的应对权衡——那是没问题的。这种情况会让你更早而不是更晚的指出你的重点。你也 能够与人交流，不用经常地担心他们不了解前因后果。

如果你不能够用三个人建造第一个版本，那么你或者需要更改人数或者需要缩减初始的版本。记住，保持你的第一个版本小而紧凑是没有问题的。你会快速的发现你的想法是否快速的进展，如果是，你会拥有一个清洁的简单的基础可以继续建造。

梅特卡夫定律(Metcalfe's Law)和项目团队

保持团队尽可能的小。梅特卡夫定律(Metcalfe's Law)，“网路的价值，为使用者的平方”，应用到项目团队的时候得到一个推论：团队的效率和团队人数的平方成反比。我开始觉得三个人对于 1.0 产品发布 是最优的...从减少你计划添加到团队的人数开始，接着减少更多。

—Marc Hedlund, 奥莱利媒体（O'Reilly Media）入住企业家(entrepreneur-in-residence)

通信流

通信在小团队比在大团队中更容易流动。如果你是项目中唯一的一人，通信是简单的。唯一的通信通路是你和客户之间。但是，随着项目人员数目的增长，通信通路的数量也随之增长。它并不是加法形式的增长，随着人员数目的增长，它是乘法形式的增长，正比于人员数目的平方。

—史蒂夫·麦克科耐尔(Steve McConnell）, Construx 软件公司（Construx Software Builders Inc.）首席软件工程师。
(摘自 《少即是多——小团队的第一生产力》，Less is More: Jumpstarting Productivity with Small Teams)

拥抱约束

让限制带领你到创新的解决方法

总是有不充足无法满足所有需要。不充足的实践。不充足的金钱。不充足的人。

这是一件好事情

不要被这些约束逼得发疯，拥抱他们。让他们指导你。约束驱动创新并强迫集中精力。不要试着移除它们，使用它们带来你的优势。

当 37signals 构建 Basecamp 的时候，我们有非常多的限制。我们有：

- 一个需要运营的设计公司
- 已有的客户工作
- 一个七小时的时差（David 在丹麦编程序，我们其余的人在美国）
- 一个小团队
- 没有外部的资金

我们感受到了“不充足“的忧伤。所以我们让我们的盘子保持小。那时我们只能够往上方这么多。我们选取大任务，把它们分解成我们一时间能够处理的小任务。我们一步一步的行动并在前进的过程中分清主次。

”不充足“强迫我们使用创新的方法解决。我们通过始终构建更少的软件减少改变的成本。我们给人们仅仅足够的特色让它们以自己的方式来解决自己独特的 问题 — 于是我们便不再是障碍。时差和空间上的距离让我们在交流中更加有效。不是人参加会议，我们的交流几乎毫无例外通过及时通讯软件和电子邮件，它们强迫我们快 速的到达重点。

约束经常是伪装的优势。忘掉风险投资，长发布周期和快速招聘。代替的是，和你目前拥有的合作。

与枯萎作斗争

曾经被称作“缓慢增长的优雅”可以被更恰当的叫做“特征枯萎病”，就像植物上的真菌一样，它节外生枝，模糊了产品的轮廓并吸干了它的汁液。特征枯萎 病的解药是，当然是，“限死的最后期限”。这导致特征被按照实现所需时间的比例被抛弃。经常出现的情况是最有用的特征需要最长的时间。于是枯萎和最后期限 的结合产生了许多我们知道和喜爱的软件，包含了充足的没有用的特征。

—Jef Raskin, 作者 (摘自 "为什么软件是它的方式")

做你自己

通过亲切友善和人性化来把自己和大公司区分开来

大量的小公司犯了试着装作大公司的错误。就好像他们意识到他们的规模是一个缺点，需要隐藏。太糟糕了。小型实际上可以是一个巨大的优势，尤其是在通讯方面。

小公司享受着更少的形式主义，更少的官僚主义，和更多的自由。**小公司天生和顾客更亲近。** 那意味着他们可以以一种更加直接和人性化的方式和顾客沟通。如果你是小公司，你可以

用熟悉的语言而不是晦涩的行话。你的网站和产品可以用一种人类的声音，而不是操着公司的腔调。小型意味着你可以和你的顾客在一起谈话，而不是居高临下的方式。

小公司在内部的交流生同样有优势。你可以摈弃形式主义。所有事情都不再需要繁杂的流程和多重的签字确认。参与流程的人都可以开放和诚实的发言。这个没有被束缚的思想流是保持小型的巨大优势。

骄傲地、无所畏惧地做到真实

虽然你可能认为，顾客可以被夸大员工数字或者你的支付能力欺骗，但是精明的，你真正希望的顾客，永远会知道真相 — 无论通过直觉还是推理。尴尬的是，我曾经参与过这样的善意谎言，所有谎言都没有带来对商业最重要的东西：和真正需要你的服务的顾客建立的有意义的、持久的 和互利互惠的关系。更好的应对应该是骄傲地、无所畏惧地对公司的实际规模和宽度做到真实。

—Khoi Vinh, Subtraction.com

不论何时

不管你在哪个行业，良好的顾客服务应该是所有客户最大的要求。我们自己对于服务是这样要求的，我们的客户又怎么会有区别？从一开始，我们就做到让客 户和我们的接触容易和明晰，不管他们有多少、甚至没有问题。在我们的网站上，我们列出了一个免费号码会转接到我们的手机；在我们的名片上，每个人都列出了 手机号码。我们向顾客强调，不管他们有什么问题，他们随时可以联系到我们。我们的顾客感谢我们这样的信任，没有人滥用过我们的服务。

—Edward Knittel, 销售和市场总监, KennelSource

什么理念才是伟大的

通过亲切友善和人性化来把自己和大公司区分开来

竭尽全力将你的软件定位在一个点上。你的软件代表的是什么？它到底是有关什么的？在你开始设计或写任何代码之前你必须清楚地知道你做这个产品的目的 — 它的前景。把理想放大些。为什么要有它？它和其他类似产品不同的地方在哪里？

这个理念会引导你的每个决定，指引你不偏离航线。任何时候有比较出格的举动时，问自己，“我们是不是还在坚守着自己的理念做事？”

你的理念必须是简洁的。应该一句话就能把想法传达到。以下是我们每个产品的理念：

- **Basecamp**: 项目管理即是沟通
- **Backpack**: 把生活中的凌乱归整
- **Campfire**: 用及时通讯软件来开展团体交流太逊了
- **Ta-da List**: 和及时贴便条做斗争
- **Writeboard**: 用不着麻烦微软的 WORD

举个例，我们 Basecamp 软件的理念是，“项目管理即是沟通”。我们强烈的感觉到对一个项目的有效沟通是引导一系列责任，参与，投资和能量的关键。它把大家统到同一个目标上来增强共识。我们清楚地知道如果 Basecamp 能做到这点，那么其他事情也就会一一水到渠成。

这个理念引导着我们尽可能地保持 Basecamp 的透明性和开发性。我们不把沟通局限在公司内部，相反我们向客户敞开。我们不考虑太多权限地问题， 相反我们更鼓励各方的参与。就是这个理念使我们放弃了图表，表单，报告，状态分析和电子表格的功能，相反的我们专注在优先问题的沟通上，如果每日新信息， 评论，该日备忘项目和文件的共享。把有关你理念的重大决定做在前面，将来其他小的决定就会变得容易多了。

白板上的哲理

有一次 Andy Hunt 和我编写一个借记卡的交易开关。有一个要点就是同一个交易不许向用户的借记卡二次收费。也就是说，不管操作过程中怎样出错，错误都只能发生在交易最终产生前，不能允许出现重复的交易。

因此，我们在共享信息的白板上用大字写下：要从客户角度出发，容许客户犯错误的可能。

还有其他大约半打多类似这样的信条，在我们创建一些复杂的产品，需要下有技巧性的决定时，这些信条给我们指引了方向。它们使我们的软件有强大的内部凝聚力和外部的统一性。

—Dave Thomas, 一个实用编程者

编写座右铭

组织需要指导原则。需要有一个纲要; 员工每天醒来时应该要知道他们为什么而工作。这个纲要最好言简意赅，富有激情：为什么你会在这里？是什么激励了你？我把这看做是座右铭 — 一段三或四个字的描述你存在的意义。

—Guy Kawasaki, 作者 (摘自 编写座右铭)

在初期时忽略细节

先粗后细

我们太过痴迷于细节。

- 两个原素之间的留白空间
- 完美的首个字母大写字形
- 完美的颜色
- 完美的用词
- 代码只能四行长，不能七行
- 90%与 89%之差
- 760px 与 750px 之差
- $39/月与$49/月之差

成功和满意来自于细节

然而，细节中并不只有成功。细节中你还会遇到停滞不前，意见不合，无数的会议和延迟。这些东西会掩煞你的信念，降低你成功的几率。

你可有常常一整天被困死在一个设计原素或一个程序代码上？可有不时觉得你今天的进展实在算不上什么真正进展？过早专注于细节就会导致这些结果。要做完美主义者有的是时间。但不是现在。

别在第一周就担心标题字体的大小。不需要在第二周就搞定什么是最佳的绿色的色调。更不用在第三周就要把“提交”按钮向右移动三个像素。先把该放的东西放上去。然后去用它。保证它是可用的。最后才去把它调整到完美。

细节是在你使用的过程中才会显露出来的。只有在使用中你才能看到什么需要进一步关注。在使用中你才会感到缺了些什么。常常走路绊倒脚你才会清楚地上什么坑洼是需要填补的。那些是当你被迫要留意的时候才需要的细节，不是一想到细节就去搞定它。

魔鬼隐藏在细节中

在选修了几堂绘画课后，我彻底摆脱了“马上投入到细节中”的态度...如果你一开始就画细节十有八九出来的画作会不怎么样。事实上，从你一开始那么做就完全错了。

你必须一开始把全局的比例分配搞对。你要从最大的一块着手，慢慢过渡到最小的。草稿必须体现模糊的主题。然后你着手润色，使整体画作具有生命力。着 色先从浅，中，深三个色调下手。这么一来你的草稿就会有明暗了。接下来，在你画作的其他部分都要秉持三个色调的应用原则。如此反复直到整体成型...

永远，都要从大到小去做。

—Patrick Lafleur, Creation Objet Inc. (摘自 Signal vs. Noise)

当问题成为问题的时候才去担心

不要把时间浪费在还未成为问题的问题

你真的的需要考虑当用户到达 10 万以上的时候会出现的问题吗？它可能已经是两年以后的事了。

如果你现在只需要三个程序员你真的有必要雇八个吗？

你难道真的马上需要 12 台高端服务器即使两台就足以让你顶一年？

就先掠过吧

人们总是预先花很多时间在还不知道会不会发生的问题上。靠，我们推出 Basecamp 的时候还不知道如何向客户收费！因为产品是月付费的，我们知道 还有 30 天的时间来搞定付费方式。我们把预先省下的时间用在解决更紧急的问题，直到产品推出后，我们才着手付费问题。结果很顺利（它迫使我们用最简单的解 决方案，没什么花哨的东西）。

别整天操心还没成型的麻烦。别过度开发一个产品。到适当的时候再添加硬件和系统软件。如果进度推迟了一两个星期，别担心，还没到世界末日。只要诚实: 解释给你的客户听，说你们正经历着成长的烦恼。他们也许不会因此无比感动，但他们起码会赞同你的坦诚。

关键是: 如果你已经掌握了你需要的信息就及时做决定。这样你就能把注意力集中到需要马上解决的问题上来。

去网罗对味的顾客

找到你产品的核心市场然后就专注进去

顾客并不总是对的。现实中你要能分辨出谁是你该针对的顾客，谁是你该放弃的。庆幸的是，互联网使得发掘有共识的顾客的过程变得无比容易。

如果你想讨好每个人那么你什么人也讨好不了。

当我们做 Basecamp 的时候，我们把市场营销集中在设计公司这块上。如此缩小市场范围，我们就更有可能吸引一些有心的顾客来成为产品的追随者。要清楚地知道你的产品是为谁推出的，集中精力去讨好这部分人。

我们最成功的一步棋

把 Campaign Monitor（市场策略观察）这个产品严格地定位在网页设计这块市场是我们走得最好得一步棋。它使我们能很容易地分辨出什么产品特色才是真正有用，更重 要地是，什么特色是该舍弃地。这样一来，我们不仅能靠瞄准一个比较小地目标市场来争取更多地客人，也因这些客人都有相近的需求使得我们的开发工作更容易 些。在 Campaign Monitor 中有大

量的功能对其他人是毫无用处的完全针对网页设计者做的。

关注在一个核心市场也便于产品的宣传。我们有了这么一个定位精准的用户群，就能知道要在他们网上经常出没的地方做广告，发布他们可能会感兴趣的文章，然后逐步建立一个产品的用户社区。

—David Greiner, 创始人, Campaign Monitor

过后才去做规模调适

你还没有必要现在就做调整

"我的应用程序能否适应万人的使用规模?"

等那真发生了再说，明白吗？如果用户的数量大大超过你的系统负荷那么恭喜！太喜欢这种麻烦事了。但在现实世界中，超大多数的网络应用程序从来都没有 到达那一步。即使你真的开始超负荷了，也不会到马上就挂了的地步。你将会有时间反应和调适。还有一点就是，只有推出产品后你才有机会采集真实的数据指标， 然后你才能用它们来推断哪些领域需要改进。

举例说明，推出的第一年我们的 Basecamp 只是在一台服务器上运作。因为这样的一个简易设置，整个实施只花了我们一周。我们并没有一开始就搞个 15 台服务器的集群或是花好几个月的时间担心规模调适的问题。

我们这么做有碰到什么问题吗? 有一些。但我们也发现大多数我们害怕的问题，比如短暂的系统滞缓，对用户来说并不是什么不得了的事。只要你及时和用户沟通，诚实地面对问题，他们是会谅解的。回头看，我们真的非常高兴当时并没有为了”完美的呈现“而把产品的发布推后数月。

开始阶段，要把建造强有力的核心产品做为首要任务，不要过分执迷于需不需要服务器组和是否有能力调整规模应变。 **先把一个伟大的产品推出，然后才去担心它无比成功了以后该怎么办的问题。** 否则你可能只是把精力，时间和金钱花在一个永远不会发生的预期上。

信不信由你，最大的问题不是规模调适，而是怎样达到你不得不需要去调适的那一刻。没有第一个麻烦哪来下一个麻烦。

反正你怎么也得回头重新审视

事实上，每个人都会有规模调整的问题，当服务人群从零到几百万的时候，所有人都必须回过头去重新审视产品设计架构的方方面面。

—Dare Obasanjo, 微软 (摘自 规模调适和创业公司)

软件要有自己的主张

你的软件应该要有倾向

一些人在论证软件应该保持中立的问题。他们说开发者限制或忽视大众诉求的软件功能是一种傲慢的表现。他们说软件应该总是能随机应变的。

我们认为那都是扯淡。伟大的软件必须要有自己的理想。伟大的软件必定是有倾向的。当人们使用软件的时候他们不只是在看功能，同时他们也在寻找一个解决方案，一种理想。决定你的理想而后追求不懈。

同时谨记，如果他们不认同你的理念的话还有无数的其他理念可供选择。没有必要总追逐你永远无法讨好的人。

一个著名的例子就是 wiki 的最初设计过程。Ward Cunningham 和他的朋友们有意把传统上认为协作文章不可或缺的许多功能都舍弃不用。他们不把每次文章的修改归功于特定哪个人，而把所有权标识都去 除了。这么一来，内容就不再自我，而成为永恒。因为他们相信重要的不是谁或什么时候写的文章。这个理念改变了一切。这个决定孕育了一个以共享己任的社区， 成为 Wikipedia（维基百科）日后的主旋律。

我们的软件走的是一条类似的路。我们的软件并不追求成为所有人的宠儿。我们的软件是有自己的性格的。他们找寻的是志同道合的用户伙伴。他们是在和有着同样理想的用户对话。你要么上来一起，要么下车。

部分，而不是残缺不全

构建一半产品，而非产品有一半缺陷

小心“所有东西除了厨房水池”的 Web 应用开发途径。投身于出现的每一个合适的点子上，你将会终结在产品的一个半傻不哏的版本上。你真正想要做的是构建一个把愚蠢一脚踢开的产品。

专注于真正必须的。好点子可以尽量坦白。**摆出产品应该成为什么样的任何点子，然后砍掉一半。**减少功能直到只剩下最必要的功能。周而复始。

对于 Basecamp，我们从 Message 开始。我们知道它是这个应用的灵魂，所以我们暂时忽略了 Milestone，Todo-list，以及其他功能。这让我们基于真实的使用情况来决定下一步怎么走，而不是凭空猜测。

从一个精简，聪明的应用开始，然后让它得到关注。就能开始在你构建的坚实基础上添砖加瓦。

无所谓

只留精髓

对于“为什么你们做这个而不做那个？”这种问题，我们青睐的回答总是“因为无所谓。”这个陈述表达了是什么让产品变得伟大。找出紧要的，略去其他。

当我们发布 Campfire 时，我们从第一次尝试此产品的人中听到下面一些问题：

“为什么只有每五分钟才有时间戳？为什么不是每一行聊天都有？ 回答：无所谓。有多少次你需要每秒或者每分钟记录谈话内容？当然不是 95%的情况下，5 分钟时间戳足够了，因为任何更多的细节都不重要。

“为什么你不允许粗体，斜体或者有色字体格式在聊天中出现？” 回答：无所谓。如果你希望强调某事，使用可信赖的 caps lock 键，或者在词语或者段落周围投放几个 * 字符。那些方法不需要额外的软件，技术支持，处理能量，或者学习曲线。除此之外，在简单的基于文本的聊天中重量级的格式无关紧要。

*“为什么你不显示当前时间房间里的总人数？”*回答：无所谓。每个人的名字被列出来，所以你知道谁在那儿，但是 12 个还是 16 个人有什么区别？如果它不改变你的行为，那么无所谓。

这些功能如果有就更好么？当然。但是他们是不可或缺的么？他们真的重要么？不是。这就是为什么我们把他们刨除在外。最好的设计师和最好的程序员不是 技能最好的，或者手指最敏捷的，或者用 Photoshop 用的神乎其神的人。他们是能够决定什么不重要的人。真正的收获源自于此。

你的大部分时间浪费在无关紧要的东西上。如果你能抛弃不重要的工作和思考，你将会获得

不可思议的生产力。

从说“不”开始

不轻易实现功能

构建部分而不是残缺不全的秘诀是说不

每一次你对一个功能说 yes 时，你正在收养一个小孩。你必须带着你的孩子通过一连串事件（例如设计，实现，测试等）。一旦这个功能出现了，你就被拖住了后腿。尽量为客户少发布一个功能，再看客户是否愤怒地离开。

不要成为 yes-man

不轻易实现每个功能。让每个功能证明自己，并且表明自己是生还者。这就像"Fight Club."。如果那些功能就像为了进来在走廊苦候了三天，你只考虑他们。

这就是为什么你从说不开始。每一个向我们提出的 — 或者我们自己提出的 — 新功能需求都遇到一个 NO 。我们倾听但是不采取行动。最初的回应是“不是现在”，如果一个需求或者功能不停的过来，我们知道才是时候对它进一步观察。那么，只是那么，我们才开始考虑实现这个功能。

当你不采纳他们的功能建议时，你如何回复他们的抱怨呢？首当其冲的是，你要提醒他们为什么他们喜欢这个应用。“你喜欢它因为我们说 NO。你喜欢它因为它没有做其他 100 件无关紧要的事情。你喜欢它因为它不试图始终讨好任何人。”

“我们不想要一千个功能”

关于 iTunes 音乐商店，Steve Jobs 私下为独立唱片制作人做了一个小型的演讲。我喜欢的瞬间是，当观众不停地举手说：“可以做[x]么？”，“你计划添加[y]么？”。最终Ｊｏｂｓ回答：“ 等等 — 放下你们的手。听着：我知道关于 iTunes 应该具有很酷的特性你有一千个主意。我们也是。但是我们不想要一千个功能。那样做很恶心。创新不是关于对每件事说 yes。而是对每一件事说 NO，除了至关重要的特性。”

—-Derek Sivers, president and programmer, CD Baby and HostBaby
(from Say NO by default)

隐藏的成本

看清功能的成本

即使一个新功能通过了对其说不的阶段，你还需要去发现它隐藏的成本。

比如，我们应该注意到功能循环（带来更多功能的功能）这种现象。我们曾经有一个需求是在 Basecamp 里加上一个“会议标签”。如果不仔细权衡这看上去 好像很简单。但是想想“会议标签”需要的不同元素：地点、时间、房间号、人员、邮件邀请、日历的整合、支持文档等等。这还不包括修改推广活动中的截图、用 户预览页、常见问题及帮助页、服务条款以及更多。在你还没搞清楚它是怎么回事之前，一个简单的想法就象滚雪球一样弄得你大伤脑筋。

对于每一个新功能你需要……

- 1. 对它说不
- 2. 强迫它证明自己的价值
- 3. 如果得到否定的答案，就此打住。如果是 yes，继续往下……
- 4. 为界面绘制草图
- 5. 设计界面
- 6. 编写代码
- 7-15. 测试，改进，测试，改进，测试，改进，测试，改进……
- 16. 检查帮助文字是否需要修改
- 17. 更新产品预览流程（如果有必要的话）
- 18. 更新用于销售的拷贝（如果有必要的话）
- 19. 更新服务条款（如果有必要的话）
- 20. 检查是否违背之前的任何许诺
- 21. 检查价格体系是否受影响
- 22. 上线
- 23. 深吸一口气

你能把握住它么？

构建你有把握的

如果你发布一个会员程序，你的系统能够处理帐户和支付问题么？可能你应该只让用户根据会员身份通过信用卡支付，而不是让他每个月撰写，签名，并且邮寄一张支票。

你能承受得起 1G 的免费空间么？还是仅仅因为 Google 这么作了？可能你应该从 100M 开始，或者只给付费用户提供空间。

底线：构建你能够掌握的产品和服务。许诺容易遵守难。确保你所作所为是在承担范围内 — 从组织，战略和财政上

人本主义

为一般概念构建软件，并且鼓励人们创建自己的解决方案

不要约束人。而是令软件宽容的接纳每个人自己的解决方案。给人们足够资源，让其通过

自己的方式解决自己的问题。然后让开路（别挡道:)）。

当我们构建 Ta-da List 的时候我们故意忽略掉了一堆东西。不分配某人一个 to-do，不标记到期时间，不对条目分类，等等。

我们保持工具干净整洁,让人们富有创造性。人们自己琢磨出了如何解决问题。如果想要添加一个日期到代办事宜项目,他们可以在该项目前添加 (至: 2006 年 4 月 7 日) 。 如果想要添加分类,也可以在该项目前添加[图书]。 理想吗?不 。 无限弹性? 是的。

如果我们试图写软件专门处理这些情景, 我们就会使它在这些担忧并不适用时的所有情况下，变得不怎么有用。

把问题的根尽力处理好，然后走开。人们将会在你的总框架内找到自己的解决方案和约定。

忘记功能需求

让你的顾客提醒你什么是最重要的

顾客想要一切在阳光下。 他们会用雪崩似的功能和特性要求淹没你。看看我们的产品论坛功能类别的要求总是盖过其它要求一大截.

我们会听到："这一点点额外的特征" ，或 "这不难办到"或"加入这个不是很简单么？ "或"仅用短短几秒钟就可以把这个加进去" ,或 "如果你加上这个，我付两倍的钱" ,等等。

当然,我们并没责怪人们提出要求。我们鼓励这样,并想听听他们怎么说。我们加入产品的几乎一切,起初都是作为客户的一个要求提出来的。但是,正如我们前面提到的,你的第一反应应该是一个 No 。 所以,你究竟应该怎样对待这些纷至沓来的要求呢? 你怎样储存它们? **你如何管理它们? 你不需要，看完之后，把它们扔掉。**

是啊,看完之后，扔掉,并且忘记它们。 听起来象亵渎了用户,但其中真正重要的会不时冒泡，提醒你。 这些都是你唯一要记住的。 这些才是根本必要的。 不必为跟踪和保留进来的每一请求而操心,让你的客户成为你的记忆仓库. 如果它真的值得一记,他们就会提醒你,直到你不能忘记。

我们是如何得出这个结论的? 当我们第一次启动 Basecamp 时，我们在 Basecamp 的一个代办事宜列表中跟踪每一个主要功能的要求 。 当一个需求被某人重复提出后,我们就用一个额外的记号更新名单上的项目(II 或 III 或 IIII 等) 。 我们计划有一天我们要检阅这份名单，并从被请求最多的功能开始依次实现之。

但事实是,我们从来没有再去看它一遍。 我们已经知道下一步需要做什么,因为我们的客户在通过重复同样的需求不断提醒我们。 没有必要留一份名单或进行太多分析,因为这一切都在实时发生。 当每天都被不停地提醒时， 你不可能忘记什么是最重要的。

另外一件事要注意:不能因为有 X 人提出需要什么,就把它列入你的产品功能。 有时不如只说不,并维持你心目中的产品。

抓住核心

问人们*不想要什么*

大多数的软件调查和研究都是围绕人们想要的产品 。"你认为有还缺失什么特征？"，"如果你可以加入一个功能,那会是多少?"， "如何使这个产品对你更有用"？

硬币的另一面会是怎么样呢? 为什么不问人们，不想要什么? "如果你可以去掉其中一个功能,那会是哪个呢?"，"你为啥不用?"， "什么让你觉得最碍事？"。

答案并不是“更多”。有时你对用户最大的优惠就是把一些东西去掉，拿出来。

创新来自说不

[创新]来自说不,否定一千件事情,以确保我们不步入歧途或是试图做得太多。我们总是在考虑进入新的市场,但是通过说不，可以让我们集中精力做那些真正很重要的事情。

—史蒂夫.乔布斯, CEO, Apple (摘自： 苹果创新的种子)

一场把软件运作起来的比赛

尽快地推出一个真实的产品

一个可运作的软件是积蓄动力，整合团队，去除行不通的点子的最佳方式。你必须从第一天开始就将它摆在首要位置。

做少一些功能，跳过一些细节，如果一些捷径能加快软件进度就大胆用，这些都是 OK 的。当你做下去的时候，你会对下一步的方向有更准确的把握。太多的故事，建模，甚至 HTML 演示都是比较虚的构想。一个运作着的软件是真实的。

只有一个真实的，可操作的软件才能拉近每个人对现实的理解和认同。避免了为一些草图和段落争得面红耳赤，最终发现这些都是无谓的。同时，你也会发现有些你想像中无关痛痒的事情事实上是很重要的。

真实的产品导致真实的行动。这才是你走向真理之路。

办实事能导致共识

当一群不一样的人开始尝试寻找和谐共鸣的时候...如果他们是一建立一个全方位的真实的产品那么他们的意见总会趋于一致。当然，如果他们只是打打草稿或是生出一些点子的话是很难达成共识的。因此，当你真正开始做一个实在的产品时，共识就比较容易达成。

—Christopher Alexander, Professor of Architecture
(摘自 Contrasting Concepts of Harmony in Architecture（对比和谐建筑中的概念）)

尽快地运作起来

我不认为我有参加过任何软件项目，不管大的或小的，是从一段漫长的规划讨论起步，不求同步发展，而又能在进度，成本或功能上成功的。把宝贵的时间浪费在发明一些不必要的或难以实施的性能上是容易的，有趣的，仅此而已，别无益处。

这个道理适用于软件开发的所有层面，“把一个产品搞起来”是一个灵活的思想。它不仅适用于一个整体的项目，微观上也适用于小规模的组件开发。

当一个可操作的组件做成后，开发者就希望知道它是否能和应用程序配合，因此他们就会尽可能快的去用它。即使一开始组件的实施并不完全，这种初期的开发协作通常会产生一个比较规范的界面和一些物尽其用的功能。

—Matt Hamer, 开发者和产品经理, Kinja 公司

冲洗一下再来过

在不断反复中工作着

别期望一开始就做得好。让软件自然成长，和软件对话。让它自然蜕变而进化。做为一个在

线的软件是不需要在完成后才推出的。设计一些界面，使用它们，分析它们，反复地做。

与其停止在把一切都事先做好做对的思路上，不如在经反复求证得出的分析判读中前行。同时，你可以更快的推出一个积极的产品，因为你并不是一味追求一出门就完美的产品。结论是是由真实世界里的反馈，真实的目标来引导你的注意重心。

反复能解脱你

如果你知道过后总是要重来一遍，你就不需追求一开始就达完美。这种明了不管如何你总是得过后重新审视一些问题的理念，能引发你先把产品想法推出去看看是否可行的激情。

可能你比我聪明

可能你比我聪明的多。

这是完全有可能的。事实上，是非常有可能。但是，如果你象大多数人的话，那么你就会象我一样，在对看不见摸不着的东西的想象方面有困难。

人类极度善于对环境周遭的事物作出反应。一只老虎走到房间里时我们会惊慌失措，灾难性的洪水过后我们懂得去清理。遗憾的是，我们在事先计划方面，在理解我们行为带来的后果方面，在重要事情的优先排序方面，却很糟糕。

或许你是少数人中能把所有事情都把握在你的脑子里的，但这也并不重要。

Web 2.0, 在这个时代我们预定每个人都已经开始使用网络，这就为一些聪明的开发者运用人类行为的不确定性创造机会。怎么说呢？就是在允许你的用户告诉你他们想法的同时，留有空间去做改进。

最后那句同时也解释了为什么你应该以这种方式开发在线软件，以怎样的方式去推广推出产品。

把你想做到的说清楚。确保各个环节无误。然后就推出和进行改进。没有哪个人自己一个能比大伙儿加起来更聪明。

—*Seth Godin, 作家/企业家*

从概念到实施

从灵感，到草稿，到 HTML，到代码

以下是我们 Get Real（求实）的过程:

脑力激荡

先要有个点子。这产品要给我们带来什么？以 Basecamp 来说, 我们是要满足自己的需要。我们想要用它来发布项目的一些更新信息。我们希望能让用户一起参与。我们知道项目都有里程碑。我们希望能有个集中归档的地方让大 家能回过头去温习一些旧的东西。我们想要有个全局观，从一定的高度来鸟瞰所有项目的进度。归结起来，这些假想和一些其他设想打下

了我们日后着手的基础。

这个阶段并不是有关一些实施的具体细节。这是一个大方向。软件需要为我们做什么？什么时候才能知道它有用？确切的说我们要做出个什么东西来？这是高阶的理念，不是像素阶段（细节）的推敲。在这个阶段，那些细节是没有意义的。

纸上草稿

草稿是迅速的，实用的和便宜的，这就恰恰是你想要开始的方式。涂些东西，画些东西，方块，圆圈，线条，什么都行。把你脑子里的想法搬到纸上。这阶段的目标是把概念转成一个界面设计的粗稿。这个阶段完全是试验性的。不存在什么答案是错误的。

创建 HTML 页面

做一个 HTML 版本的功能界面（或一个区间界面或流程界面，如果这么做更合适的话）。发布一个实在的东西，这样一来大家就都可以看到它出现在屏幕上的样子。

以 Basecamp 而言，我们先做“发布一条信息”的界面，然后是“编辑信息”的界面，然后一步步下去。

先别写任何程序代码。只把 HTML 和 CSS 的框架搞出来。有关细节实施是后面的事。

上代码编程

当模型框架看起来过得去又兼具一些足够必要的功能时，就是开始上代码编程的时候了。

在这整个过程中要记住保持机动弹性，要有多次反复的思想准备。应该随时有这个意识：舍弃某些已完成的步骤重新来过，如果成品看起来丑陋不堪。数次重复这个过程是很自然的。

远离设置首选项

要帮你的客户决定一些小处细节

假设你将面临一个困难抉择：在一个页面上可以发布多少条信息？你的第一反应可能是，“不如做个设置首选项，在那里人们可以选择 25，50，或 100 条每页”。这么做可是一个方便自己之门。你必须要自己做一个决定。

设置首选项是一种逃避困难抉择的方式

你不是运用你的专业去决定最佳的选择，相反地把问题留给了客户。表面看起来好像是你在帮客户的忙，事实上你只是会使他们更忙（客户自己已经是够忙的 了）。对客户而言，面对无穷无尽的设置选项是一个很令人头痛的问题，不是一件好事。客户不应该去烦恼细枝末节 — 当是你的责任的时候就不要让别人去担待。

设置选项也是邪恶的因为他们使软件变得冗余。更多的选项就需要更多的编程代码。而且你还要花额外的时间在测试和设计上。还有很多选项排序和显示界面等你可能从来没见过的东

西。这意味着隐藏的软件瑕疵：破碎的布局，凌乱的表格，奇奇怪怪的页面排序问题等等。

你要拿主意

替你的客户下简单的决定。这也是我们在 Basecamp 上用到的诀窍。每页可发布信息数是 25 条。项目总览页显示最近的 25 条信息。信息反时序排（最新的在上面）。最新近的 5 个项目会显示在控制面板上。不需要任何设置选项。它本来就该这样。

是的，你有可能下了一个不太好的决断。没什么大不了。如果事情发生了，人们会抱怨，会让你知道。照样，你可以做调整。Getting Real（求真求实）说的就都是有关能够一路做灵活修改的道理。

做设置首选项是要付出代价的

事实证明，加设置首选项是有代价的。当然，有些首选项也有重要的作用 — 并且可能是关键的页面职能。但每提供一个选项都有不菲的代价，你应当仔细考量其价值。很多的用户和开发者都没能理解这个道理，最终付出很大成本，宝贵的资 本只带来一点点的价值...我发现，如果你是信奉要靠设计优秀默认功能而不是懒惰地去添加设置首选项的人，那么自然而然地你的总体 UI（用户界面）会走上 正确的道路。

—Havoc Pennington, 首席技术指导, Red Hat (from Free software and good user interfaces （自由软件和优秀用户界面）)

"搞定!"

决定都是暂时的，那么拿定主意就继续到下一步

搞定。现在就开始把它看成一个有魔力的词。当你到达“搞定”的阶段就表明你已完成某事。一个决定已经下了，走下一步。“搞定”也表明你已经聚集了能量。

慢着，如果你搞砸了，下了一个错误的决定怎么办？没问题。 **这并不是什么开颅手术，它是一个在线应用程序。** 我们一再强调，在开发过程中你总是需要不时回过头去调整软件的功能及想法。不管你计划得多周密总有可能一半左右的东西没做好。所以，不要做“到死都要调查分析”的傻事。那样做只会慢了进度和磨去意志。

相反地，要知道以“朝前看向前走”为重。要跟上拿主意的节拍。做一个迅速简单的决断，如果它行不通那就再回头修改。

要接受多数决断都是暂时的有时效性的现实。要接受错误必将发生的现实，同时也要认识到这并不是什么大不了的，只要你能迅速改正之。执行，积蓄能量，而后前行。

做一个执行者

当我听说有人对自己的点子很具保护性时觉得很可笑。（那些在告诉我一些简单的概念之前希望我签定保密协定的人。）

对我而言，如果不去执行的话点子是一无用处的。它们只是倍数。执行才是价值万金的。

理由:

- 糟糕的点子 = -1
- 脆弱的点子 = 1
- 普通的点子 = 5
- 好点子 = 10
- 伟大的点子 = 15
- 超闪亮的点子 = 20

- 没有执行 = $1
- 柔弱的执行 = $1000
- 普通的执行 = $10,000
- 好的执行 = $100,000
- 伟大的执行 = $1,000,000
- 超强的执行 = $10,000,000

如果要成就一番事业，你必须将二者相乘。

最闪亮的点子，如果没有执行，最多值$20。如果它乘以优秀的执行，那么就值$20,000,000。

那就是为什么我不爱听他人的点子。只有当看到它被确实执行下去了我才有兴趣。

—Derek Sivers， 总裁，程序员， CD Baby 公司 and HostBaby 公司

放飞去让大众测试

在现实使用中测试你的软件

让真人在真实的环境中使用你的软件，这是无可代替的。取得真实的数据。取得真实的反馈。然后在那些信息的基础上进行改进。

常规的可行性测试太死板了。实验室的设置并不能反应现实。如果你站在别人的背后观察监视，你可能多少了解一个方案是否行得通，但人们普遍在摄像机面前无法自然表现。当被别人这么看时，大家都会很小心地不去犯错 — 但错误却正是你所要获知的信息。

相反，在正式软件中发放 beta 功能给一些有选择的用户。让他们能同时使用 beta 功能和已发布的功能。这样这些测试的功能就能曝露在真实的数据和流程中。从这你就能取得真实的数据。

另外，不要搞正式版和 beta 版的游戏。两者不应该有区别。另外做一个 beta 版本只会得到一个轻描淡写的试用。正式版本，注入一些 beta 的功能，才能得到全方位的体验。

Beta 书的发布

如果开发者在发布代码的时候都很紧张，那么书的发行商和作者在推出 一本书时岂不得吓死。当一部书付诸印刷时，要做修改是一件无比麻烦的事。（事实上并非如此，但这些和旧技术联系在一起的感觉和记忆还弥漫在行业中。）因 此，发行商总是花很大的功夫（和成本）要在书发行前争取把书做到“对”为止。

当我写 Agile Web Development With Rails（用 Rail 做前卫的网页开发）这本书时，很多的开发者有相当明确的需求：我们现在就需要这书 — 我们想要知道更多有关 Rails。我也可能从出版商的角度出发。“它还没完成”，我会这么说。但来自社区压力和 David Heinemeier Hansson 的督促使我改变了想法。我们在书最后完成前两个月以 pdf 的格式预先发布了这书。结果是令人难以置信的。我们不仅卖了很多书，而且得到了反 馈 — 许多的反馈意见。我开发了一个自动化系统来攫取读者发布的看法，最终得到差不多 850 个有关错字和技术错误的报告，另有添加新内容的建议。所有这些的解决 方案都浓入到最后的书面出版中。

这是一个双赢的局面：我能提交一个完善了的纸面出版书籍，社区也能及早地得到他们需要的内容。如果你是在一场赛跑竞争中，早些把作品交出去可以争取更多的追随者而不是过后引来更多的竞争对手。

—Dave Thomas, The Pragmatic Programmers（《实干的编程者》）

要快

- 1. 决定它是否值得做，如果是的话:
- 2. 尽快去做 — 不需完美，只需做下去
- 3. 保存。上传。发布。
- 4. 看人们的反应

> 虽然我并不总爱给产品加新功能，但一旦那个值得去做的"yeah!"时刻到来，新的功能一般几个小时后就能上到网页上去，有瑕疵但就这么发布了，让用户反馈来引导下一步的修补工作。
>
> *—Derek Sivers, 总裁，程序员，CD Baby 公司 和 HostBaby 公司*

缩短你的时间

把它分块来做

做几周甚至几个月的预期是不现实的。事实上你无法预见那么远的将来会发生什么状况。

所以，缩短你的时间范围。把一个时间段分成一个个小块。把一个 12 周项目看成是 12 个周项目。与其去推演一个要花 30 个工作小时的任务，不如把它们分成更现实的 6-10 个小时的小任务。然后一块一块地去执行。

这个理论同样适用与其它问题。你是否有碰到一个很大的问题想都想不过来？把它划分开来想。就这么一直把问题分成小块及更小块直到你能消化它为止。

小一些的任务和时间表

软件开发者是一群特殊的乐观主义物种：面对放在他们面前的编程任务时，他们总会想，“那不难！花不了多少时间。”

所以，如果给一个程序员 3 周去完成一个大型任务，她会花两周半拖拉，然后用一周的时间在编程上。这种不按期执行结果就会造成和预期任务要求脱节，因为每个任务总是会比表面看起来更复杂。还有，谁还记得三周前整个团队达成的详细共识是什么？

给程序员一个下午去编一个小的特定的模块，她就会有办法把它赶出来，然后准备进入到下一个任务。

小一些的任务和时间表比较好管理，可以省去一些可能由于繁多产生的误解，同时你改变主意或重新做的成本也会较小。小一些的时间表可以督促开发者，让 他们更有机会去享受某种成就感，同时不让他们有更多的理由去想，“哦，我还有很多时间去做那个项目。现在让我给我 iTunes 宝库里的歌曲评评级先。”

—Gina Trapani, 网页开发者，编辑 Lifehacker, the productivity and software guide（效率和软件指南）

事实的真相

下次如果有人硬要你回答一个答案尚未可知的问题 — 不管它是有关一个截止日，一个项目的最终成本，或填满 Grand Canyon（大峡谷）需要多少牛奶 —这类不发生无法知道的问题，你尽管可以从避免空谈的角度出发：说“我不知道”。

这不仅远不会毁坏你的信用，同时能显示你对下决定的慎重和用心。不要只说些听起来精明的话。你还需平衡游戏规则，将问题重整成有利协同合作的对话。通过对你的预期所能达到的效果的逐步明确化的讨论，你就能和其他人一起打造一个共识的平台，揭开数字背后的真相。

—Merlin Mann, 创造者和编辑， 43folders.com

解决冲着你来的那个问题

近来我的网络记忆中最自豪的一件事就是发布和引进"nofollow（译者注，一个 HTML 属性值）"的态度。没人事先讨论过它。没有一大堆的会议 座谈可以让一些无聊人用来进行其含义和编程本质的争论。没有征求意见的过程，于是不需要把一个简单的理念做成得花上 20 行 xml 的小程序（还得花时间解读 如何用这个程序，最终结果就是不用它，因为我不清楚是否设定在版本 0.3 或 3.3b）。

它简单，有效，只提供选项给需要的人 — 这对于网络中不在乎技术规格的框框条条或觉得遵礼节太费时的一群，无疑是非常重要的。

有时，解决后来的 20 个难题往往不如搞定直冲你来的那一个问题来得有用和审慎。它不仅仅是一个战胜滥码的一个小胜利（所有和冗余代码的斗争胜利都不会是大的），而且是对那些热爱简洁和迅速的结果并以之为己任的网页开发者的一个大胜仗。

—Andre Torrez, 程序员和副总工程师， Federated Media Publishing

一致性

拒绝分隔

很多公司将设计，开发，广告撰写，支持和营销分隔成不同的战斗单位。虽然专业化有它的好处，但是它创作的环境却让员工只看到自己的小世界而不是 web 应用的整个背景。

尽你所能的，整合你的团队，这样才能有一个健康的，反复的讨论贯穿整个流程。建立一个制约平衡的系统。不要让事情在翻译中迷失。让广告撰写者和设计者一起工作。支持的疑问一定要让开发者看到。

更好的情况是，雇用拥有多项天赋的人，他们可以在开发过程中担任不同的角色。最终的结果是一个更加协调的产品。

独处的时间

为了让事情做好，人们需要不被打扰的时间

37signals 跨越 4 个城市和 8 个时区。从犹他州的普罗沃到代码的哥本哈根。我们五个人分隔了 8 个小时。这 8 个小时的距离的一个积极的副作用是独处的时间。

一天中只有四到五个小时我们都醒着并在一起工作。其他的时间，美国团队在睡觉而 David，人在丹麦，在工作。剩下的时间，我们在工作而 David 在睡觉。这让我们一天中一半在一起而另一半独处。

猜猜在一天中哪一部分完成我们完成的工作最多？独处的那部分。这没有什么惊奇的。很多人喜欢的工作时间是清晨或者午夜 — 他们不会被打扰的时间。

如果你有很长时间没有被打扰，你就能渐入佳境。这个情境是你最有生产力的时间；这个时间内你不必在各种各样的任务中切换思维；这个时间你不会受到干扰，比如回答问题或者是查找东西，发送邮件或者应答即时通讯。独处的时区是产生真正进展的地方。

渐入佳境需要时间。这就是为什么干扰是你的敌人。这就像深度睡眠 — 你并不直接进入深度睡眠，你先进入浅睡眠，然后你会逐渐进入深度睡眠。任何干扰都会让你从头开始。**深度睡眠是真正的睡眠魔法发生的地方；独处的时间是真正的开发魔法发生的地方。**

在工作中建立一条规定：一天中一半的时间作为独处时间。从上午 10 点到下午两点，任何人都不可以和别人谈话（除了午餐时间）。或者让一天的前一半或者后一半作为独处时间。只要保证这个范围是连续的，为了避免破坏生产力的干扰。

成功的独处时间意味着赶走交流痴迷。在独处时间中，放弃即时通讯，电话呼叫和会议。不要打开随时更新的 email 程序。只需闭上嘴去干活。（Just shut up and get to work.）

进入最佳状态

我们都知道知识工作者在稳定状态工作最出色，这种状态也被称作“渐入佳境”，他们全神

贯注于工作，开足马力，忘了周围的环境。他们忘了时间的流逝， 在绝对的集中精力下产生了巨大的成果...那番在于这种情境太容易被破坏。噪声，电话呼叫，吃午餐，需要开车 5 分钟去星巴克喝咖啡还有合作者的打扰 — 特别是合作者的打扰 — 都破坏了这个情境。如果你需要花一份钟处理合作者问问题的干扰，这足以破坏你的集中经历，那么你需要花费一个小时重新达到有效率，你的总生产力变得很糟糕。

—Joel Spolsky, 软件开发者, Fog Creek Software
(摘自 这些人从哪里得到他们(非原创的) 思想?)

会议有毒

不要会议

你真的需要会议吗？会议经常出现在概念不够清楚的时候。不要求助于会议，试着简化概念，于是你可以快速的讨论它，通过电子邮件，即时通讯或者 Campfire。目标就是避免会议。你避免花费在会议上的每一分钟是你真正做事情的每一分钟。

对于生产力的毒性没有比会议更厉害的了。以下是几点原因：

- 他们将工作日分解成小的，不连续的片段从而打乱了你的日常工作流程
- 他们经常是关于词语或者抽象概念，并非真实的事物（比如代码片段或者一些接口设计）
- 他们经常每分钟传达非常小的信息量
- 他们通常包含至少一个笨蛋，轮到他时不可避免的通过无意义的话浪费大家的时间
- 他们偏离主题比大雪中的芝加哥出租车还容易
- 他们的议程经常非常模糊以致于没有人真正的明确他们的目的 T
- 他们经常需要精心的准备但是人们无论怎样罕能做到

在你*确实地必须* 开会（这必须是一个少见的事情）的时候 , 坚持这些简单的原则：

- 设定一个 30 分钟的计时器。当它响的时候，会议结束。句号。
- 邀请尽可能少的人。
- 没有明确议程的时候不要开会。

少开会

有太多的会议了。放弃那些没有意义的和没有效果的会议。只有当需要讨论重要的商务议题的时候或者你需要建议，赞同或者一致意见的时候才召开会议。即便如此，不要不加选择的邀请人 — 不要不必要的浪费人们的时间。

—Lisa Haneberg, 作者 (摘自 不要让会议统治你！)

分解它

当子昂目增长的时候，增加人员带来了减少的回报。最有趣的原因之一就是增加的交流通道的数量。两个人只需要相互沟通；只有一条简单的交流途径。三个人有三条交流途径；4 个人有 6 条。事实上，交流途径的增长是指数级的……很快，备忘录和会议汇占据整个工作时

间。

解决方法是明显的：把团队分解成小的，自治的和独立的小单位以减少这些交流途径。

详细的，把程序也分解成更小的单位。既然问题的一大部分起源于相互的依赖（全局变量，函数间传送的数据，共享的硬件等等），找一个方法分割程序以消灭— 或者减少— 个体间的相互依赖。

—*The Ganssle Group (from Keep It Small)*

寻找和庆祝小的胜利

每天发行点什么

软件开发中最重要的就是激励。激励是局部的 -- 如果你没有被你正在做的事所激励，机会就没有应有的那么好。 实际上，很有可能变得更糟糕。

冗长，滞后的发布周期是激励的杀手。 这使得庆祝活动之间间隔太长的时间。 另一方面， 可以庆祝的迅速胜利是极大的激励动因。如果你让冗长的发布周期压碎迅速的胜利，就杀死了激励。 并且这样也会杀死你的产品。

所以， 如果你的发布周期是在一个月以内， 拿出每周一天（或者没 2 周拿出一天）对取得的小胜利进行庆祝。 并扪心自问： “我们可以在 4 个小时内发布些什么？” ， 然后去实现它。这可以是

- ***一个新的简单特性**
- ***一个对现有特性的小改进**
- ***为了减低支持负担，重写一下帮助信息**
- ***去除那些并不需要的表格项**

当你发现这些 4 小时的迅速胜利时， 你将发现庆祝的理由。 这样鼓舞了士气，增强了激励 并且 再次肯定了团队正在向着正确的方向前进。

不需过早招聘太多员工

慢慢加人迅速发展

初期后期都并不一定要壮大队伍。即使你接触过 100 个顶级人才，一口气把他们全招来也并不是什么好主意。没有办法能让这么多人迅速的融入到统一的企业文化中去。你将遭遇令人头痛的人员培训、性格不和、沟通不畅、发展方向不同等诸多麻烦。

所以不要随便招人。真的。不要招人，另想办法。让你陷入烦恼的这件事是真正必要的吗？你不做又会如何呢？能不能用某种软件或者改变做事方法来解决呢？

ge 前执行总裁 Jack Welch 每次裁掉一个人之后并不会马上招人来顶替他的位置。他想看看在那个职位和人员空缺的情况下能支撑多久。我们当然不主张用裁人来验证这个理论，但是我们的确认为 Jack 的做法有一定道理：你并不需要你考虑中的那么多人手。

如果没别的办法再考虑招人。但是你应该清楚知道你需要什么样的人，怎么向他介绍工作任务，以及具体要他负责解决什么样的棘手问题。

Brooks 的原则

给延期的软件开发项目添加人手只会更加拖延进度。

—Fred Brooks

编程与莫扎特的安魂曲

一个优秀的程序员在完成单个工作任务时不存在因沟通和分工而产生的额外开销。而五个程序员坐到一起完成同一个任务的时候必须分工合作，那将花费很多 时间……用很多一般的程序员而不是几个足够好的程序员将产生的真正问题在于：无论让他们干上多久，也绝对没有优秀程序员干得出色。五个 Antonio Salieris 也永远写不出莫扎特的安魂曲，哪怕给他们 100 年的时间。

—Joel Spolsky, Fog Creek Software 的软件开发人员 (摘自 Hitting the High Notes)

摸底

先和候选员工在测试项目中协作

看作品、简历、例程、工作经历是一码事，和他在一起工作是另外一码事。只要有条件，应该和准团队成员一起去“试试车”。

在聘用人之前，我们会给他们一个小项目琢磨琢磨。我们能从中看出他们怎么管理这个项目，他们怎样进行沟通，他们具体怎么做等等。和他们一起设计或者编写几屏代码能看出很多东西。你能迅速摸清和他们是否心有灵犀。

这种事规划起来比较难，但即使只能拿出 20 或者 40 小时来做也比没有强。适合不适合都能

看得很清楚。如果不适合，先摸清情况能给双方避免很多麻烦和风险。

小处着手

从分派一个小的测试任务开始。不要一股脑把你的工作任务都拿出来。给你的新虚拟助理一两个测试项目来做，看看化学作用如何发生。开始的时候，大家很容易在和和气气的氛围中忽略掉潜在的问题。记住这只是在试车。

—Suzanne Falter-Barns, 作家和创意专家
(摘自 How To Find And Keep The Perfect VA)

行胜于言

根据对开源社区的贡献选择潜在的技术人才

典型的通过学历、简历等方式来招聘技术人员在很多方面都是很愚蠢的。应聘者毕业于什么学校、学习成绩如何真的那么重要吗？一份简历或介绍信真能信得过吗？

开源社区是为那些需要招聘技术人员的人准备的礼物。通过开源社区，你可以在很长的时间跨度里跟踪某人的成果和贡献，无论好坏。

这意味着你能以他做过什么而不是说过什么来判断他是否合适。你可以通过考察真正重要的方面来做决定：

- **工作质量**
  很多程序员说的时候口若悬河，实际去做的时候却错漏百出。通过开源社区，你可以直观的了解这个人的编程技巧和素养。
- **文化视角**
  编程就是做判断。很多很多的判断。判断力遵循于这个人的文化水平、价值观和观念。考察候选人在编码、测试和社区讨论（争吵）中作出的具体决定，看看他在文化上是否和你有共同点。如果没有共同点，每个决定都将引发一场争论。
- **热情程度**
  根据定义，参与到开源项目至少是需要一些热情的。不然为什么他要在屏幕前浪费业余时间？对开源项目的投入程度就能看出候选人对编程的热衷程度。
- **执行力**
  如果完不成任务，再聪明的头脑、再合适的文化倾向和再高的热情也带不来有价值的软件。不幸的是，很多程序员都做不到这点。应该去找那些热忱工作的人。要做比买卖的时候，需要有这样的人——这个人要把货带出门，而且他也愿意去做。
- **社会经验**
  和人一起共事很长时间，一起经历过紧张和悠闲，一起经历过起起落落，才能看出他们的真实性格。如果他缺乏教养，没有社交技巧，把他排除掉。

说到程序员，我们只聘用通过开源社区认识的人。我们认为其他任何方式都是不负责任的。我们聘用 Jamis 是因为我们关注过他在 Ruby 社区的参与程度和发布成果。他在前文所述的各个方面都做得很好。不需要次要原因，我们只评判真正重要的——他的工作质量。

不用担心团队成员“课外活动”的活跃度带走他们的注意力和工作热情。有句老话是这么讲

的：如果你想做好一件事，去找你所认识的最忙的人。Jamis 和 David 两个都是对 Rails 社区有重大贡献的人，同时，他俩还驾驭着 37signals 的技术部门。热爱编程并且能干活的人就是你真正需要的团队成员。

对开源社区的热情

你最希望从新员工身上看到的就是他对自己工作任务的热情，而最能看出这点的办法就是在开源项目中去追溯他的提交记录。

—Jarkko Laine, 软件开发人员
(from Reduce the risk, hire from open source)

寻找全面发展的人

选择能快速学习的多面手，而不是专攻一面的专家。

我们从来不会用一个信息架构师。那实在太狭隘了。对于我们这样的小团队，招技术层面那么窄的人没用。

小团队需要能扮演不同角色的人。你需要会编程的设计师。你需要懂设计的程序员。每个人都应该对怎么构建信息（无论它指的是什么）有自己的想法。每个人都应该思路清晰。每个人都需要能和客户打交道。

而且每个人都需要能”在路上换档“。要记住小团队经常需要迅速改变前进方向。你需要有人能持续的调整和学习，而不是固步自封，只会干一件事。

热情是装不出来的

选择快乐的和技术水平中等的，而不是令人不满的专家

热情，是无法伪装的。招人的时候，不要认为你需要一个技术大师或者技术名流。他们往往自以为是。一个水平说得过去的快乐员工胜过于愁眉苦脸的专家。

寻找充满热情的人；寻找你信任他可以独立完成任务的人；寻找在发展缓慢的大公司受过折磨，并且渴望新环境的人；寻找为一起去建造你正在建造的东西而感到激动的人；寻找对你所厌恶的事物同样感到厌恶的人；寻找为入你的伙而感到兴奋不已的人。

为提问加分

留意候选人是否对你的项目提出很多疑问。热心的程序员总是想尽量去理解存在很多疑点的问题并快速提出可能的解决办法和改进方案。阐明问题能产生一种 认识：项目的做法很多，但基本上取决于你对你的 Web 应用如何运作的想象。当你深入到细节，你能感觉到这个人是否在文化水平上符合。

—Eric Stephens, BuildV1.com

炼字师

招文字功底好的人

如果你在琢磨从几个人选中挑出哪一个来填补空缺，选文字功底好的那位。无论他是设计师、程序员、市场人员、销售人员还是其他，写作技巧都很有用。简洁高效的写作和文字编辑能力可以带来简洁高效的代码、设计、邮件、即时信息以及更多。

这是因为要当好写手并不只是堆砌词藻。好写手懂得沟通的技巧，他们让事情易于理解，他们能站在别人的立场考虑问题，他们知道什么是可以忽略的，他们思路清晰。而这正是你需要的才能。

条理清晰的头脑

好的写作风格是头脑清晰的指示器，清晰的头脑能有条絮地梳理信息和论点，还能帮助（而不是逼着）其他人去理解。这种技巧能渗透到代码的编写、口头表达、即时通信（对于那些远程协作的人来说），甚至更深层次的职业素养和可靠性等领域。

—Dustin J. Mitchell, 开发人员 (摘自 Signal vs. Noise)

有清晰的文字才有清晰的思路

清晰的文字能带来清晰的思路。表达它的时候你才知道你对它了解些什么。好的写作风格也从一定程度上反映出人的性格。让读者省事，而不是给自己省事。

—Michael A. Covington, 佐治亚州立大学计算机科学教授
(摘自 How to Write More Clearly, Think More Clearly, and Learn Complex Material More Easily)

界面先行

开始编程之前先设计界面

很多应用人们在开始做的时候都抱着先编程的心态——那是个坏主意。编程是构建应用的过程中最笨重的部分，也就是说，它改动起来最复杂，成本也最高。做应用应该始于界面设计。

界面设计相对来说是比较轻量级的。一纸草图修改起来简单，成本也低。html 页面的设计修改起来或是推翻也比较简单。修改程序就不是这么回事了。界面先行可以让你保持灵活；先行编程则会让你陷入花费额外开销的圈套。

界面设计先行的另一个原因是——界面就是你的产品。你向人们销售的产品正是他们能看到的。如果你最后才推出界面，就会出现缺口。

我们先从界面开始，所以从一开始我们就知道这个应用看上去如何，给人感觉怎样。在开发过程中，界面将会不断的改进。合理吗？易用吗？是不是解决了手里的问题？这些问题只有你和真实的界面打交道的时候才能回答。设计优先让你保持灵活而且让你能更早地回答那些问题。

开创 Blinksale 的橙色钢笔

当我意识到现有的帐单管理软件都无法让我满意的时候，我决心把我想要的帐单管理解决方案画出来。我拿出一支橙色钢笔，因为它是那晚上唯一好用的东 西，然后我用了几小时画出了 75%的用户界面。我把它拿给我妻子 Rachel 看，当时她正在烫衣服，我问她：“你觉得怎么样？”她微笑着回答：“你真该把 它实实在在的做出来。”

接下来的两个星期里我重新斟酌了设计，并且做好了当时可能成为 Blinksale 第一版的大部分静态页面。除了那些用橙色钢笔画的草图，我们从来没 有做过其它网格之类的东西。而且直接进入 html 页面的设计让我们一直为项目变得现实起来而感到激动，即使在当时我们并不知道我们正在陷入什么当中。

html 页面一完成，我们就和开发员 Scott 一起商量关于 Blinksale 的想法。已经设计好的用户界面在好些层面都发挥了非常好的作用。首 先，它让 Scott 能亲眼看到，使他为我们的目标感到激动。它远远不止是一个创意，它是真实的。其次，它帮助我们可以精确衡量 Scott 需要花多长时间和 精力才能把设计变成能跑的应用。当你在对一个项目的孵化提供资金上的支持，越早能做出预算越好。界面设计成为了我们在项目初期的评估基准。最后，用户界面的设计就像一个向导，能在我们进一步开发的时候提醒我们这个应用程序的核心目标。当我们临时决定添加新功能的时候，我们不能简单地说：“好的，那就加 吧！”我们必须回到设计上来，问自己新功能应该放在哪，如果没有它的位置，它就不该加。

—Josh Williams, Blinksale 创始人

震中设计

始于页面的核心然后向外延展

震中设计就是关注于页面的本质——震中，然后再向外拓展。这意味着，一开始要忽略细枝末节的东西：导航条或者导航标签、页脚、用色、边栏、标识等，从震中着手，先设计页面中最重要的内容。

页面赖以生存的是其核心。举例来说，如果你正在设计显示博客文章的页面，那么核心就是博客文章本身。不是在边栏里的文章分类，不是顶部的页头，不是底部的评论提交表单，而是实际的博客文章单元。没有博客文章单元，这就不是一个博客文章页。

只有当这个单元完成之后你才能开始考虑页面中次重要的元素。次重要的元素完成之后，你再转战第三重要的元素，以此类推。这就是震中设计。

震中设计规避了传统中 “先搭建框架，再填充内容”的方式。在那种方式里，人们先建立好页面布局，然后把导航条包含进去，然后插入有关销售推广方面的东西，到最后才把核心功能——页面的实际意义所在用来填充剩下的空间。这是本末倒置的做法。

震中设计让你从一开始就关注于真正重要的部分。先做必需要的，再做其他的。结果是给用户一个更友好、重点清晰的界面。并且，这样可以让你马上和设计、开发人员展开对话而不是等到页面所有部分都各就各位之后。

考虑三种情况下的解决方案

常规、初始、错误三种情况下的设计

对于每一个界面，你需要考虑可能出现的三种情况：

- **常规**
  一切运行正常的话，人们看到的是一个充满内容的界面。
- **初始**
  人们第一次使用这个应用，在加入内容前的界面。
- **错误**
  有错误发生时，人们看到的界面。

常规的情况众人皆知，它将花去你最多的时间。但别忘了也要为初始和错误两种情况花些时间（接下来的文章作更详细地说明）。

初始界面

期待一个周到的初次运行体验

忽略初始界面的阶段是你会犯的最大错误之一。初始界面是应用留给人们的第一印象，永远不会有第二次这样的机会……这个么，你应该知道。

问题是，设计界面时，通常是事先用数据填充起来。设计者总是用临时数据填满页面模板，每一个列表、文章、输入框、边边角角里都填上内容。这时候界面看上去很棒。

然而，产品在初始状态下是没有内容的。当新人注册，他们将从初始界面开始。就像是一个博客，用户需要自己去充实它。而在用户输入文章内容、链接、评论、日期、边栏的信息或其它数据前，整体外观无法成形。

不幸的是，用户会在初始界面时决定产品是否值得一用——在这个内容和信息最少的阶段来判断产品的适用性。如果你设计的初始界面有缺陷，人们就不知道缺少些什么，因为他们感觉什么都没有。

然而大多数设计者和开发人员仍然认为这种状况理所当然。他们没有很多时间为初始界面做设计，因为当他们开发和使用产品的时候，里面填满了他们用来测 试的数据。他们甚至没遭遇过初始界面。当然，他们可能也以新用户的身份登录过几次，不过他们主要的时间是沉浸在一个充满数据的产品里。

一个有用的初始界面应该包含些什么？

- 利用它作为添加新手指南和热门推荐的机会。
- 给出一个填满内容的页面截图，这样能让人们有所期待。
- 讲解如何上手，页面最终会像什么样子等。
- 回答第一次来的访客会问到的关键问题：这是什么页面？我在干什么？页面有内容的时候是什么样子？
- 做好预期准备，帮助减少挫折感、恐惧感和大的迷惑。

第一印象太关键了。如果没有设计出周到的初始界面，会为你的产品或服务留下反面的（错误的）印象。

没有第二次机会……

我 觉得苹果 OS x 操作系统的界面中另一个深受 Steven Jobs 影响的是安装和初次运行的体验。我想 Jobs 一定深深理解到第一印象的重要性……也许 Jobs 在考虑初次运行的体验时会想，它可能是一个用户对电 脑上千次用户体验中的一次，但是它将是最重要的千分之一，因为它是一千次中的第一次，人们对它寄予了期望并形成初步印象。

—John Gruber, 撰稿人和开发人员 (摘自 Interview with John Gruber)

做好防御

出错时的设计

我们得承认：在线上的程序总有出错的情况。无论应用设计得多么用心，无论做了多少测试，用户仍然会遇到问题。那么如何处理这些不可避免的故障呢？做好保护性设计。

保护性设计就像是在小心驾驶。和司机在行车时必须留意道路是否打滑、鲁莽的司机以及其它危险情况一样，网站建设者们必须不断寻找会造成访问者困惑和不满的故障点。好的保护性设计能决定用户体验的好坏。

关于保护性设计的内容我们可以单独写成一本书。实际上，我们已经写了。这本叫《网站的保护性设计》的书对于想学习如何改进出错界面和其他故障点的人来说是非常好的资源。

记住：你的应用可能90%的时间都运行良好。但是如果在用户需要帮助时置之不理，他们是不会忘记这一点的。

应用环境胜过一致性

这里有意义的那里不一定有意义

动作是使用 按钮 还是用 链接 来实现 ？这取决于那个动作。一个日历视图是应该使用列表方式还是表格方式实现？
这取决于 它要用在哪里 并且 要显示多长的时间。全局的导航链接是否需要出现在每个页面上 ？是否每个地方都需要一个全局的搜索条 ？是否在每一页上需要一个完全相同的页脚？答案是“这取决于应用环境”

这就是应用环境比一致性重要的原因。 如果你的设计在那种状况下有意义，不一致也没有什么大不了的。 只给人们重要的。 给他们所需要的，并且去掉其不需要的。比保持一致性更好的是保持正确。

聪明的不一致

一致性不是必不可少的。很多年来，学习用户界面和用户交互的学生都这样被教导，界面中的一致性是界面设计中的核心守则之一。也许这在软件中成立，但是在Web上，这不对。Web上真正重要的是，在每个独立页面，用户是否能够快速、简单地前进到流程中的下一个步骤。

在“好的创新”（Creative Good），我们把这叫做“聪明的不一致”：确保流程的每个页面都给用户提供他在流程中的那个位置所真正需要的东西。只因为了和网站的其它部分保持一致，加入多余的导航因素，是很愚蠢的。

-- 马克.赫斯特 ， Creative Good 和 Goovite.com 的创建人

拷贝书写是界面设计

每个字母都重要

拷贝书写是界面设计，伟大的界面是写出来的。如果你认真考虑每个像素，每个图标，每个字体，那么你要相信每个字母都是至关重要的。当你写界面时，要常常要设身处地的为阅读你的界面的人着想。他们需要了解什么？你怎样才能简洁明了阐述之 ？

你标注一个按钮的名字是 提交 还是 保存 或者 更新 还是 新建 或 创建？这就是拷贝书写。你是写 3 句话还是 5 句？是用一般的举例说明，还是详细阐述？标注内容是 新增的 还是 更新的 或者是 最新更新 还是 更正的 ？是否有新消息：是用 5 还是 5 个新消息 或者是 5 个 抑或是 5 个 或者 5 个消息还是 5 个回复 ？所有这些都很重要。

你也需要和你的读者讲同样的语言。只因为你写的是一个 Web 应用，这并不意味着你可以使用技术行话。想想你的客户，想想这些按钮和词语对其意味何如。不要使用缩写 或者 大多数人不懂的词语。不要使用内部隐语。不要听起来象一个工程师和另一个工程师的谈话。保持简短和亲切。说你要说的，别说废话。

好的书写就是好的设计。界面用词和设计要相辅相成，这很少有例外。图标要有名字，表单项要有示例，按钮要有标签，流程要有一步一步的指示，退款政策也要有一个清楚的说明。

统一界面

合并管理功能到公共界面

管理界面—就是那些用来管理选项，人员等的屏幕界面。—看起来很低劣。这是因为大多数时间都花在了对公共界面的设计上。

为了避免这种低劣管理界面的并发症，不要分开去开发系统管理界面。而要，把管理功能（例如，编辑，增加，删除）嵌入到应用界面开发中。

如果你必须维护 2 套分开的界面（如：一个一般用户用，另一个管理员用），这 2 个都会很难受。实际情况是，你把它们搅在一起来，就像同一个税你交了 2 次。你强迫自己重复并且这还意味着事情变麻烦的机会大大增加。

拒绝分离的界面

应用软件就是全部。所有可以被改变的，增加的 或者调整的，都应该直接通过应用的管理区域完成。这使得我们能精确感知我们客户的需要，从而通过解决他们的问题和疑问来帮助他们。我们的客户也不必担心，必须登入分开的界面来完成不同的工作。一分钟前，他们也许在处理和客户的约会，随后一分钟他们也许要添加新雇员。他们不会为了在不同的应用间跳来跳去而烦恼，通过一个统一的界面，他们能够很快的适应这个应用软件。

—爱德华. 尼特，KennelSource 销售和市场部主任

更精简的软件

使代码尽量简化

2 倍的代码只会是你的软件复杂 2 倍。实际上，**每当你增加代码量，软件的复杂程度就成指数化的上升。**每个较小的增加部分，每个改变，每个内部依赖，和每个选项都有串联的效果。鲁莽的增加代码，在你意识到这之前，你将创造一个可怕的巨大泥巴球。

战胜这种复杂性的方法就是要更精简的软件，这意味着更少的特性，更少的代码和更少的浪费。

关键是要把一个需要很多编码的困难问题，转换为一个需要较少编码的简单问题。 你可能没有精确的解决这个问题，但是这没有什么。只花 20%的努力就解决了原来问题的 80%就是一个大胜利。 原来的问题再糟也不会耗费我们 5 倍的努力去解决它。

更简化的软件，意味着你把水晶球先放一边。 暂时不要去预测未来的问题，先处理好今天的问题。为什么？你对明天的担心很少会变为现实。不要为去解决这些幻想的问题而使自己陷入泥潭，效率下降。

从一开始,我们设计产品就围绕着精简软件这一概念.只要有可能,我们就把难题分解成简单的部分. 我们发现这些简单的部分不但容易实现，而且易于实施和支持,他们更容易理解也很容易使用.所有这些，正是我们与竞争对手的很大区别;我们作产品不是去作更 多，而是去作更少。

- *简化的软件比较容易管理.
- *简化的软件意味着精简你的代码库
- *更少的复杂维护作业(快乐的员工) .
- *简化的软件降低你变化的成本,使你能很快适应变化.您可以改变主意，并且无需改变很多的代码.
- *简化的软件较少错误.
- *简化的软件意味着较少的支持.

在软件中选择包括或省略那些特性，和简化软件有很大关系.不要担心拒绝那些很难实现的特性 .除非他们绝对必要的,去掉这些，从而节省时间/努力/迷惑.

要慢下来.对一个想法不采取行动,过一个礼拜,当初期的嗡嗡声在耳边渐渐平息时，看看它似乎仍是一个很好的想法。 额外深入思考时间往往会帮助你的大脑想出一个容易的解决方案。

鼓励程序员的相反提议

你想听到: "你的建议将花 12 小时.但我有个办法只需一小时。就是别作 x ,作 y 。" 让软件往前推.告诉程序员以他们认为最好的方式去努力奋斗。

同时,寻求避免多写软件的招数.你可以向客户建议其它的替代路线，能否通过改变屏幕上的拷贝,而不需要改变软件模型?举例来说,你能通过规定上传图片的具体尺寸,而避免进行在服务器端的图像操作么?

对于融入你的应用程序的每个特性,问自己:是否有一个不增加软件代码量，就实现这个的方法? 只写你需要写的代码，绝不多写.结果你的应用将更加精简和健康.

没有什么代码比没有代码更灵活!

良好的软件设计的"秘密" 不在于知道把什么加入代码;而是要知道把什么拿出来！ 正是要识别出 硬点和软点,并且知道那里应该留下空白/空间,而不是试图去填满更多设计.

—Brad Appleton, 软件工程师
（摘自： 没有什么代码比没有代码更灵活! ）

复杂性和大小不以线性规模扩展

最重要的软件工程法则也是最不为人所知:复杂性和大小不以线性规模扩展… … 一个2000 行的程序比其一半大小的程序需要两倍多的开发时间.
—The Ganssle Group *(摘自：保持小巧)*

为了快乐而优化

选择使团队兴奋和倍感激励的工具

快乐的程序员是多产的程序员。这就是我们为了快乐而优化的原因，你也应该这样做。不要只从行业标准中，或者是从衡量性能的角度去挑选工具和作业模式。看一看无形的东西：有没有激情，自豪和技能这些感觉？你真的为在这种环境下一天工作 8 小时，而真心的感到快乐？

这在选择一种编程语言时，有其重要。 别管公众对之的对立感觉，它们不是生而平等的。尽管任何一种语言可以创建任何一个应用，但是正确的选择不仅可以使你的努力更易实现 或者让你更好过，而且它能使你心情愉悦，精神鼓舞。这都是一些使日常工作充满乐趣的小细节。

快乐有级联的效应。快乐的程序员作正确的事。他们编写简单，可读的代码。他们采用干净，有意义的，可读的、优雅的方法。他们自得其乐。

我们发现 用 Ruby 语言编程简直太有福了，并且把这种幸福传递到了使用 Rails 框架的其他开发者身上。两者的使命宣言都是要为 人 和他们的快乐 而进行优化。

总结一下, 你的团队要使用他们喜爱的工具来工作。我们在这里是以程序设计语言的观

点来讨论，但是这个观念是普遍正确的，无论是对应用系统，平台软件，还是其他任何事。选择能够点燃激情的导火索。你就能保证激情和动力，并且收获一个更好的产品。

你需要的那种工程师

我使用 Ruby on Rails 来构建我们的应用的众多理由中的一个是，它的特质是如此的优雅，高产出和设计优美。它倾向于吸引那些特别关心这些特质的工程师... 这样的人正是你的团队所需要的，因为他们创造你需要的这种美丽的，优雅的，高产的软件，来赢得市场。

---- 查理.朱雷， Nisus 软件 行政主任（摘自： 信号 vs 噪音）

代码说话

倾听当代码推动时

倾听你的代码。它会提建议。它要推动你。它将告诉你缺陷在哪里。它将建议做事的新途径。它将帮助你坚持精简代码的模式。

有一个新的功能需要花费好几个星期和写几千行代码来开发么？这是你的代码在告诉你兴许有个更好的办法。是否有一个简单的途径能在 1 小时之内编码搞定某个功能，而无需采用一个复杂的方法而花费 10 个小时？再一次，这是代码在指引。注意倾听。

代码可以引导你找到那些又便宜又轻便的修正方法。注意，当一个容易的途径出现时。当然，那个容易实现的功能也许和你当初想的不完全一样，但那又如何？ 如果它工作的很好并且使你 有更多的时间去做其它事，这是一个守门员。

仔细倾听

不要担心设计，如果你仔细听你的代码，一个好的设计就会浮现... 倾听技术人员的想法。如果他们抱怨很难进行改变，那么你要重视这些抱怨，并且给他们时间去搞定。

— 马丁. 福勒 ， ThoughtWorks 首席科学家（摘自： 设计完了吗？ ）

如果付钱给程序员去砍掉代码...

如果付钱给程序员去从软件中砍掉代码，而不是编写新代码。软件就会好得多。

—尼古拉斯.那个如珀特 ， *MIT 媒体技术教授* （*摘自：*（AIGA 会议）故事的剩余部分）

管理欠债

付清你的代码和设计的帐单

提到债务，我们往往把它和钱联系起来，不过债务也来自其它的形式。你会很容易的积累起来编码和设计上的债务。

一起砍掉一些功能性的坏代码，不过还是有些小危险积攒下来，变成了债务。整个扔掉一个设计，这很好但不是真正的好，因为你已经又重作了一遍。

时不时的发生这种情况很正常。实际上，这往往是一个可以帮助你快速完成编码所必需的技巧。但是你必须认清你的债务，并在恰当的时机进行偿还——通过清除坏代码并重新设计那些不好不坏的页面。

你还应该定期撇开某些要交税的收入，偿还你的代码，设计债务。否则，你就得付利息（修补裂痕），而不是直接付清本金（继续前进）。

打开门

通过 RSS、API 引入数据

不要试图把客户隔绝在外。当他们需要的时候，就能够按照他们的方式得到信息。为了做到这一点，就不得不放弃对数据的密封。而是让数据以本来的方式显现。通 过 RSS feeds 的方式使别人可以访问信息，为第三方开发者提供 API 接口，使他们可以在你的基础上进行构建。当你这样做的时候，对客户来说生活会更加便利，并 且扩展了你的应用程序的能力。

人们常认为 RSS feeds 只适合跟踪博客或新闻站点，其实它的能力远不只此。它还能够使用户不断跟踪应用程序的变化，而不需要反复登录。通过使用 Basecamp feeds，用户可以一边阅读新闻，同时留意项目消息、待完成的工作项和关键节点，而不需要不停地在网站上登录。

API 使开发者能够以较小的代价为你的应用程序开发增值产品。例如，Backpack 提供了一个 API，Chipt Productions 用它可以在 Mac 操作系统下生成一个 Dashboard 小程序。这个小程序可以让用户从桌面增加和编辑提示符、列表项。用户对这个 小程序大加赞赏，有些甚至认为这是吸引他们使用 Backpack 的关键因素。

其他提供免费的数据，并使自己从中受益的好的例子：

* **Google Maps API** 提供大量有趣的小物件，使用户可以从其他数据源提取信息（例如 公寓清单）并标绘在地图上。

* Linkrolls 提供一种方式，可以使人们从他们的网站上获得最新的 **del.icio.us** 的书签。

* **Flickr** 允许其他公司访问其商业 api，从而使用户可以购买相片集、海报、DVD 和邮票。“目的是将它完全开放，当人们处理照片时可以得到最多种的选择。” Flickr 公司的 Stewart Butterfield 如是说。

不同凡响的小程序（Widget）

当 37signals 发布了 Backpack，我的第一印象是...嗯

因此当 Chipt Productions 发布了小程序 Backpack widget for Tiger 的时候，——简直太酷了，我马上就下载并试用了——，当我再次打量 Backpack，结果呢？印象完全改观。

现在，每当我的脑子里冒出一个念头，我就弹出这个小程序，从键盘输入，提交——完成了。邮件来了，要检查邮件？弹出小程序，从键盘输入，提交——完成了。 这个小程序是一个直接的头脑堆栈，在我使用的每一个 Mac 机上都唾手可得。由于所有的部分都基于 Web，因此不存在版本控制或同步问题——只是内容的输入 流，完全不用考虑它们流向哪里以及以后我讲如何访问它们。

—Todd Dominey, Dominey Design 的创建人 (摘自： 试用 Backpack)

功能定义一点用都没有

不要写功能定义文档

这些蓝图文档通常和成品几乎毫无关系。理由如下：

功能定义文档是虚幻的

功能定义文档不反映真实情况。一个应用只有在被构造时、被设计时，和被使用时才是真实的。功能定义文档只是纸上谈兵

功能定义文档是无关痛痒的

功能定义文档可以用来让人感受到参与的乐趣，措辞温和但是并不是那么有用。它们不关心艰难的抉择，不关心成本——而这些正是建立一个成功的应用所必须考虑的。

功能定义文档只能达成虚幻的共识

看文字并不能让人们达成共识。大家可以读到同样的文字内容，但每个人的想法却可能不同。以后将不可避免地发生这种情况：“等下，我不是那样想的。”“啊？我们可不是这么说的。”“是的，就是这样的，我们大家都同意了——你还签过字呢。”我相信，你知道该怎么做。

功能定义文档逼迫你在拥有最少资讯时作出最重要的决定

当你刚开始构建时，你知道的是最少的。而做得越多，用得越多，你才能了解得越多。这才是你应该做出决定的时候——即当你有足够多信息，而非信息少的时候。

功能定义导致功能泛滥

*功能定义阶段对整个过程没有什么推动。*写点东西加个标注，看上去并不需要什么代价。你可以加上他们欣赏的功能，让那些令人头疼的人高兴。然后你按照那些写下来的文字标注设计，而不是为人设计。最终你得到的将是一个拥有 30 个栏目的臃肿站点

功能定义让你无法变通、变化和重新评估

一个功能一旦明确下来，得到认同，即使在开发阶段你就意识到它是个坏主意，你也不得不照做。一旦你开始做某事，一切都在变化，而功能定义却不会去处理这些实际情况。

那么，你应该做什么去替代功能定义呢？ 去写一个简明的替代文档，以此引导你去做那些真正的事。 写一页的说明去描述这个应用要做什么。 使用平实的语言并且要简短。 如果要阐述的内容超过了一页纸，就太复杂了。 这个过程不应该超过一天。

接下来开始建立界面----界面将成为功能文档的替代物。 在纸上简单快速地画些草图。然后把它写成 html 代码。 界面设计是每个人都会认同的共同基础，这不同于，大段的文字

可以有不同的理解。

人人都使用同样的屏幕界面时， 混乱不见了。在你开始担心后台代码之前，要建立一个人人都可以看得见的，使用的，点击访问的，并且可以“感受到的” 界面。 一定要尽量把自己置于客户体验之前。

忘记那些锁定的功能定义。 它们迫使你做大，在太早的时候逼你作关键决策。 略过功能定义阶段，你将可以便于改变并且保持灵活性。

没用的功能定义

“功能定义”几近无用。 我还从来没有见过一个足够全面和足够准确的功能定义文档。

而且，我见过大量的基于功能定义（文档）的无用功。 这真是写软件的唯一最坏途径，这从它的定义就可以看出： 为了教条而写软件，而不是现实。

—Linus Torvalds, Linux 之父 (摘自： Linux: Linus 谈规格书)

和阻碍作斗争

我发现人们常常坚持在任何设计工作开始之前，要先准备大量的需求文档， 这真是一些“阻碍”，使整个过程变慢。（这些人通常对设计没有任何的贡献和创新思维）

我们所有的最佳工作都是这样做出来的， 我们把头脑中的一些关于站点改进的想法， 先作一个（静态）的快速原型， 再改动一点设计，然后使用真实数据建立一个活的原型。 在把原型上的一些累赘剔除以后，我们通常都会得到一个健康运转的项目，并且取得很好的成果。

—Mark Gallagher, 公司内部网研发 (摘自： 信号 vs 噪音)

别写死文档

根除不必要的文书工作

避免写功能规格定义是一个好的开始,但不要仅只于此；要处处避免过分的写文档工作。除非有个文件确实要演变成现实, 否则别写它。

建造出来,别写出来。如果你需要解释什么,先建造一个原型而不是写一份冗长的文档。实际的界面或原型是你正在构建一个真正的产品的很好说明。另一方面,一纸文档,只能说明它们终将被丢入垃圾桶。

这里有一个例子:如果一个线框文件是注定要停止,并且永远也不会直接变为实际设计,不要为此烦恼。而如果线框开始作为一个线框造型并且演变为实际设计,那就用它。

脱离实际应用而存在的文档是没用的。 它们对你没有帮助。 你在作的所有事都应该最终变为真实存在的。 如果一个文档在变为真实之前，就停止了，那它完蛋了。

谁也不会去读它

我从不会去数在我们研发队伍的身边有多少这样无聊的，未读的，沾满灰尘的，有很多页的产品规格书或者业务需求文档。 我们只管去编码，讨论问题，质疑和进行用户测试。 我也和那些写了好多冗长的，描述性的电子邮件或者编码规范文档的 研发人员一起工作过，同样那些文档也没人读过。

开发 Web 应用并不会因为有丰富的文档就会一帆风顺。 软件开发是一个不断变化的，反复迭代过程，这其中包含着 相互作用， 快速决策 和一堆在前进路上不断遇到的无法预测的问题。 这些没有一个是能够，也不应该在文档中捕获的。

不要浪费时间去堆砌这些冗长成册的不实文档； 没人去读它。 如果你给你的产品足够的空间去成长，到最后这个产品无论如何也不象你写到的那样，这个事实应该是你得到安慰。

—吉娜 特怕尼, Lifehacker,的研发和编辑 生产力和软件指南

告诉我一个短故事

写故事,别写细节

如果你发现你确实需要来解释一个新的特征或概念,写一个简短的故事说明之. 别陷入技术或设计的细节,只讲一个短故事. 象你在正常的交谈时和一个人讲话的方式一样。

它并不需要成为一个短文。只要象记流水账似说明发生什么事。如果拿出你正在开发的屏幕作背景，来简述这个故事,那就更好了。

坚持经验,而不是越来越拘于细节。考虑战略,而不是战术。一旦你开始建立的你的那部分应用，战术自然就会适得其所。现在你只是要想出一个故事,发起讨论,然后使你步入正轨。

使用平实的语言

填入真实的文字而不是测试用的胡乱用语

测试用的胡乱文字（Lorem ipsum dolor）一直是设计者的忠实伙伴。 虚拟文字帮助人们认清设计完成后的观感。但是这些虚拟文字会很危险。

测试用的胡乱文字改变了拷贝的视像。 使得文本内容弱化为一个视觉设计元素 --- 文本形状 ---而不是其本来面目： 有价值的某人录入/或要读的信息。 虚拟文本意味着你不会看到

当真实信息录入后出现的不可避免的改变。 这意味着你不会知道在你的站点填写表格时会是什么样子。 虚拟文字是隔在你和现实之间的面纱。

你需要真实的拷贝去认识某个字段要多长才合适； 你需要真实的拷贝去懂得表格是怎样扩展或收缩的； 你需要真实的拷贝去感知你的应用看起来究竟怎么样。

只要有可能，你就要使用真实，准确的用语。 如果你的站点或应用需要数据输入，那就录入真正的交易数据。 并且要敲入实际的文本--不要从其他来源拷贝粘贴。 如果那是一个姓名，就敲入一个真实的姓名。 如果是个城市， 录入一个真正的城市。 如果是一个密码，就要重复 2 次，录入 2 次。

确实，从上到下过一遍表格并把每项都填入垃圾数据（"asdsadklja" "123usadfjasld" "snaxn2q9e7"）是很简单。但是这不真实。 你的客户一定不会这样使用。 当客户被强迫要长途跋涉时，你却走捷径，这能算是精明么？ 当你用连发开火的方式快速录入假数据时，你根本不会体会到填表格时的真实感受。

象你的客户那样去做，你才能更好的理解他们。 当你更好的理解他们时， 你会感同身受，做出更好的界面。

胡乱用语垃圾

由于没有想象力想出真实内容“大概”应该怎样， 一项设计考虑因此丢失。“这里该是文本”使得含义晦涩， 因为没谁意识到这些文本是要实际被读到的，明白易懂性大打折扣。 机会也会浪费，因为这些你使用的胡乱用语垃圾不是真实内容，你无法从中悟出机会。这些文本的作用很小，因为，不能这样用，我们兴许也可以创建一堆可爱的空 格。

—汤姆 史密斯, 设计和研发 (摘自： 我憎恨 Lorem Ipsum 和 Lorem Ipsum 使用者)

个性化你的产品

你的产品的个性类型是什么？

把你的产品想象成一个人。 你要赋予他什么个性类型？ 有礼貌？ 严肃的？ 慈悲的？ 精确的？ 有趣的？ 无表情的？ 认真的？ 自由的？ 你想它以怎样的面目示人，是偏执的还是令人信服的？ 是作为一个万事通？ 抑或是谦逊并且人见人爱 ？

一旦你确定下来，就要在构建产品的过程中时刻记得保持这些个性特征。利用这些个性指导拷贝写作，界面设计 和 功能项配置。 一旦你想要改变什么， 自问一下这会不会改变

你的应用的个性特征。

你的产品有声音---会一天 24 小时的不停地与你的客户交谈。

免费样品

免费赠送

外面的世界很嘈杂。为了让人们在喧嚣中注意到你，免费赠送一些东西吧。

聪明的公司都知道免费发赠品是吸引顾客的一个好方法。看看苹果公司，他们提供免费的 iTunes 软件，是为了建立客户对 iPod 和 iTunes 音乐存储的需求。在网下的世界里，零售商们也这么做。星巴克公司认为，每送出 5 个饮料赠品，就刺激了一种新商品的购买。不要太小气。

对于我们来说，Writeboard 和 Ta-da list 是完全免费的应用，用来吸引人们逐渐使用我们的其他产品。同时，对于我们所有的应用，我们总是提供一些免费的版本。

我们希望人们来体验我们已经完成的产品、界面和有用的东西。一旦他们着迷了，就更有可能升级到一个付费产品（付费产品提供更多的项目或页面，让人们获得更多的功能，比如：文件上传和 ssl 数据加密）。

可吃尺寸的块

做成可吃尺寸的块： 专门设计的，较小的功能可以促使客户来试用。 至少要把一个产品或服务再细分成很小的可吃尺寸的块，这些小块的功能花钱不多，简单或者充满乐趣。

—本 麦克康奈尔 和 Jackie huba, 客户博客的教堂 的作者 (摘自： 什么是客户的福音布道?)

拿出你的最热单曲

设想一下在你的每个专辑中拿出一首单曲作为免费奖励，让全世界下载 --- 就像电影的预告片 --- 就像送往电台的最热单曲---使得人们想去买你的音乐。

不要担心这首曲子被盗版。 让人们去演奏， 去拷贝 ， 去共享 ，去分发。要有信心，一旦全世界都听过，他们会付钱买更多。
—Derek Sivers, 董事长 兼 程序员, CD Baby and HostBaby (摘自： 免费促销单曲)

来去自如

让注册和注销的过程毫不费力

在你的程序里注册和注销应该尽可能简单。

如果我是一个客户，想用你的产品，这应该是一个毫不费力、轻松容易的事。在销售站点的每一页都放上一个大大的、清楚的注册按钮。告诉大家这是多么容易的事：“从注册到登录仅仅只需要 1 分钟！”

应该总是有个自由选项，让客户不用输入信用卡信息就能进行产品演示。有些公司需要用户确认、预约或者特殊密码才能体验他们的产品，根本就没有必要这么做。任何人随时都可以自由地体验我们的产品而无须提供任何信息。

注册表单要尽可能短。不要问一些并不需要的问题，不要抛出一个长得吓人的表来为难大家。

这些原则同样适用于注销过程。永远不要指望把人们“困在”你的产品里。当有人决定注销他们的 Basecamp 帐户时，尽管我们感到遗憾，但是我们不会让注销过程繁琐或是含混不清。个人帐户页就有一个“注销我的帐户”的链接，并不需要发邮件、填写特殊表格或者是回答问题。

同时，如果他们决定离开，要确保能导出他们的数据。我们确保客户随时可以轻易地导出 xml 格式的所有信息和评论。那是他们自己的数据，他们理应能按自己的意愿来处理。

这一点很重要。因为给予人们掌握他们自己信息的能力可以塑造对我们的信任。这给了他们一座通向自己的数据岛的桥梁。如果他们找到了一个能提供更优服务的地方，让他们自由离开。这么做是对的，而且可以建立良好的声誉。

轻松离开

不要勉强留住用户。如果他们要离开，就让他们带上在你网站创造出来的全部内容，然后自由离去。必须敞开粮仓，集中精力留住客户，所以他们愿意回来，而不是因为被门卡住了才不得不回来。

—Charlie O'Donnell， 分析师， Union Square Ventures
(摘自 10 Steps to a Hugely Successful Web 2.0 Company)

Silly Rabbit, Tricks are for Kids

规避长期合同和注册费用等

谁都不会喜欢长期条款，提前终止费或是一次性的安装费，所以要避免这么做。我们的产品付费方式为月付，不用签订条款，你可以随时取消，而且从不会有什么安装费。

别企图去找什么“高明”的方式以赚得更多的钱。Earn it.

塑料子弹

用提前通知和保留条款来缓和坏消息给用户带来的打击

要发布类似提价这样的坏消息啦？应该多次提前通知，尽量把坏消息带给用户的痛苦降到最低。同时，应考虑在保留条款中规定，现有客户在一定时间内仍可以原价使用产品。用户就是你的奶油和面包，你要让他们感受到被重视，而不是被欺骗。

好莱坞运作

从挑逗 到 预演 到 开幕

如果你的应用选在森林中发布，每人会去使用， 它弄出动静了么？ 关键一点是，如果你在发布应用之前没有一些事前吹嘘，人们根本就不知道有这回事。

为了闹出个大动静和引起期待， 学学好莱坞风格的运作： 1） 挑逗 ， 2） 预演， 3） 上演

挑逗

提前几个月要开始不断透露些暗示。 让人们知道你在干什么。 发布一个徽标。 在你的博客中发布一下进展。 保持神秘，但是要播下种子。同样，建立一个网站，好让你可以从那些感兴趣的人那里收集电子邮件。

这个阶段， 你也要开始引诱专业和业内人士。 这些家伙站在前沿。 他们是引领时尚潮流的人。 他们的虚荣心和其作为时尚领导者的地位,使其容易被吸引。 告诉他们，要安排他们参加秘密进行的内部预演。 如果有一个象 Boing Boing， Slashdot 或者 Digg 这样的站点链接了你的（Web）应用，你就会有大量的浏览访问量跟进。 另外，你在 Google 的网页评级也会水涨船高。

预演

发布前的几周， 开始预演特征。 给人们幕后入口。 描述产品的主题。 对于 Basecamp ，我们发布了屏幕截图 和 高亮度的提示条， 里程碑 和其他一些特性。

同样， 告诉人们此应用背后的思想和原则。例如 Backpack ， 我们在上线前公布了产品宣言。 这使得人们去思考并且谈论这个应用。

你也可以给少数人发放一些特殊“金元券”，让他们可以提早开始使用这个应用。 这些自我感觉提前被选中，享受了特殊荣耀的人其实充当了你的 beta 版测试人员，你从中获益。

同理， 一旦上线发布，鼓励人们来注册，你因此可以得到大量的电子邮件用来发起闪电般的宣传攻势。 当我们的应用正式发布时，成千上万的电邮向我们涌来，这对赚取眼球帮了大忙。

上线（开演）

正式上线时间到了。 现在人们可以真正地去“剧院”看你的应用了。 发邮件给那些注册的

用户。 发布你的全面营销网站。 尽力到处散布你的信条。 让博客都链接到你。 公布你的进步： 已注册了多少用户？已经进行了那些更新/调整 ？ 显示你的成长势头并且保持住。

通往发布之日的道路

当我们刚一得知 Blinksale ，真的要有所动作时，我们就开始在我们的邮件列表中散播诱人的小花絮。 那些列表用户都曾经申请要收到我们项目的最新消息， 他们是我们的"粉丝"，听我们的。 如果你已有权和一组人对话，这些人就是你使用这种策略的最佳地方。

我们做的第二件事就是取得和更多人宣扬我们产品的权利。 在 Blinksale 上线前 6 周，我们在站点上加入了一个花絮页面，宣称更简便的在线寄送发票方式即将诞生。 这个页面给出了刚刚好足够的信息，以引起兴奋和悬念，而没有泄露那些需要保密的敏感细节。 在页面的显著位置上有一个时事快讯订阅表，不要别的，只有填入电子邮件（保持简单），就可以使那些有兴趣的用户在产品发布时得到通知。

我 们同样也给将近一打左右的，可能对此产品有兴趣的朋友和同事传了话， 他们也开始把花絮页面上的内容，通过他们的博客和网站作了进一步的宣传。 短短几天之内， 我们的邮件列表中就有了上千的订户。 这些都是及其重要的人---他们在要求多了解我们的产品 并且他们想知道我们何时（上线）发布。

终 于， 我们在上线前的 2 周， 邀请了少数的朋友，同事 和业内专家，来进行发布前 beta 测试一下 Blinksale 产品。 这让我们把产品拿到用户面前，通过使用进行检验，我们感到这会使我们从中受益。他们还可以帮我们，在产品发布时作宣传。这点值得强调，因为我们没有强迫谁 去使用，或者去写评论文章。 我们只是想让产品面市，并且想让人们在产品发布时谈论它。 最后， 如果你要这样引起共鸣，你最好确定你的产品可以交付。 否则， 就象光打雷不下雨。（有云不下雨）

当上线之时， 我们向邮件列表发了封信， 通知我们的博客朋友 ，并鼓励我们的 Beta 测试者畅所欲言。 令我们高兴的是，努力取得了巨大回报。 上线发布后不一会儿， 成千上万的人访问了我们（应用）的网站，其中几千用户注册了使用我们的产品。

—Josh Williams, Blinksale 创始人

强大的推广网站

从 花絮 到 预演 到 上线

最佳推广工具是一个伟大的产品 。 全世界会发现你拥有一个确实有用的应用

即便如此， 你仍需要一个顶级的推广站点。 在这个网站中要有什么？ 这是一些主意：

- **概览**: 说明你的应用及其益处
- **导游**:引导人们体验各项特性
- **屏幕截图和录像**: 展示应用面貌并演示如何使用
- **宣言**: 阐述其背后的哲学和思想
- **案例研究**: 展示现实生活中的案例的可能性
- **共鸣**: 引用来自客户，评论，新闻的证明材料
- **论坛**: 提供社区会员相互帮助的场所
- **费用和注册**: 尽快让人们使用你的应用
- **博客**: 博客提供新闻，技巧等，使你的网站保持活力

驾驭博客波浪

博客可以比广告更具效力，（而且便宜很多）

广告很贵。 而且评估各种类型的广告的效果，已被证明比广告本身还有昂贵。 当你没有财力和时间去走传统的广告路径时，应该考虑一下通过博客推广的路线。

应该一开始就建立一个博客，不仅可以招揽顾客，还可以提供有帮助的建议，技巧，花招，链接等等。 我们的 信号 vs 噪音 博客，由于每天都张贴出一些 有帮助的，启迪的 和 有趣的信息和奇闻轶事，每周都会吸引来成千上万的读者。

因此，当要推广我们的第一个产品 Basecamp 时。 我们在 SVN 上透露了这个消息，并开始传播。 象 强森 考特克，BoingBoing 用户，吉姆 库都 这些人，和身份各异的其他一些拥有博客的人，帮我们提升了可见度。推广工作起飞了。

Ta-da 代办事宜列表是另一个显示了基于博客进行市场营销威力的伟大案例。 我们在 信号 vs 噪音 上只发了一个关于 Ta-da 上线的帖子， 几周后就有 200 多个博客提到了它，并且 12000 人注册了自己的 Ta-da 待办事宜列表 帐户。 关于 Backpack 的消息传的更迅速。 上线后 24 小时， 就有 10000 个用户注册。

尽早征集

尽早获得共鸣和注册

我们已经略微谈到这点，并且认为值得重复： 要建立某类网站 , 要尽快征集邮件。选定你的域名并且发布你的徽标，还要写上一两句，或者至少也要暗示一下，你的应用是作什么用的。 然后，要求人们留下电子邮件地址。现在，你已步入正轨，因为已经有了一帮想在上线发布时准备得到通知的人。

通过教育推广

与世界分享知识

当一个老师作为选手出现在 Jeopardy 电视节目上时，亚历克.特别可 常常评论说，教师是个高尚的职业。因此可以肯定和别人分享知识是十分美妙和很有收获的一件事。 他是对的。 **当你教的主题是你的应用时 ，这具有双重目的：你能回馈社区，这不仅支持了你，还可以在同一时间 为你的那些漂亮的宣传曝光取得加分。**

作为一项推广技术，教育是用一种软的方式来 把你的名字---你的产品的名字----摆到更多的人面前。而不是采用强硬推销 "购买此产品"的方法，通过提供有价值的服务，你越来越引起重视。此举激发了积极的共鸣，是传统的营销策略不能比拟的。那些受你教育的人将成为你的福音传 播者。

教育可以有许多形式。人们通过在你的网站上发表窍门希望同他人分享。在会议上发言并且会后和与会者见面。举办讲习班，让好奇的粉丝可以学到更多并且与你恳切交谈。 接受出版物的采访。写文章，分享有益的资料。并且写书出版。 ;）

如我们历来采用的例子，使用“黄色渐淡技术”，这是我们发明的一种方法，用来巧妙地高亮标明页面上最近修改过的区域。 关于它，我们在“信号与噪音”写了一个帖子。这个帖子不断被浏览，已经被浏览了成千上万次（至今应该已有极大的访问量）。

那个帖子在教育和推广层面都起了作用。学了一招，并且原先很多绝不会知道我们的产品的人也被暴露在产品面前。 另一个例子：在我们的使用 Ruby on Rails 研发的阶段，我们决定把基础框架开放源代码。这被证明是个明智之举。我们把一些成果回馈社区，建立商誉，赢得我们团队的认可，收到了有益的反 馈，并开始收到来自世界各地的程序员的补丁和无私奉献。

教学总是好事。 你先付出。你在帮助别人,就会得到一些很好的促销效果。你甚至可以沐浴在一丝高尚的荣耀里。那么，你知道世界想要听到什么？

先付出

我们博客的文章和技巧是我们网站中最受欢迎的部分。通过传授电子邮件行销，确保我们的顾客从我们的软件中获益最大化。如果他们能提供更好的服务给顾客，那么他们就可能得到更多生意，而这反过来又为我们创造了更多生意----多赢。

自由地分享我们的知识，还帮助我们树立了作为业内专家形象，并加强了我们与现有客户的合作关系。 他们知道我们关心他们工作的质量。最后，我们从与读者分 享我们文章的搜索引擎和博客那里，得到了大量的流入访问浏览量。 如果我们不写那些文章，这些人们永远也不会听说我们的软件。

—David Greiner, 运作监理 创始人

特色食品

他们渴望得到的,就给他们

新的或有趣的特性是一为你的应用引起共鸣的很好的办法。特殊兴趣小组爱咀嚼"特色食品"，并且吐回回馈社区。 好吧，这是一种不愉快的比喻，但你得明白这一点。

举例来说，通过使用 Ruby on rails，一个新的开发框架，我们的 basecamp 引起了开发社区的巨大关注。

在我们应用中使用的 ajax 的元素引起许多反响，甚至连 “商业 2.0 杂志” 也把 37sigals 作为一个"Ajax 的关键角色"， 把我们与一些响当当的公司，如：谷歌，雅虎，微软以及亚马逊等并列。

另一个例子：很多博客注意到了 Basecamp 的 rss 支持，因为它是企业的 rss 的最早的应用之一 。

集成 iCal，一个看似轻微的小特色，使我们得到了原本永远也不会提到我们的应用的 Mac(苹果)相关站点的大量关注。

小团队时刻准备把新想法整合到软件中去。而大公司要处理官僚主义瓶颈，你能够迅速实施新想法并且因此受到重视。

掌握使用最新时髦技术的花边噱头，是一种有效且廉价的方式来引起轰动效应。那种不要加入最新暧昧技术的说法只是为了引入注意。但是如果你正在使用了一些新的或引入瞩目的技术，一定要继续使用并且把它在特殊兴趣社区中大肆宣传。

（译注：iCal 摘自： Apple 中国- Mac OS X - iCal
iCal 允许你以天，星期或者月查看你的计划表。并且使用 iCal 强大的搜索功能，找到你日程表上的内容就更容易了。）

跟踪你的日志

研究日志并跟踪共鸣

你必须知道谁在谈论你。检查你的日志，找出共鸣来自哪里。谁链接了你？谁在诽谤你？

哪些列在 technorati blogdex， feedster， del.icio.us 和 Daypop 上的关于你的博客文章最热门。

找出来这些博客，然后让感觉到你的存在。 在这些博客上留下评论。感谢他们链接到你。问他们，是否想被列入你的特别提前通知名单，这样他们将最先知道你的应用的未来发布，更新等等。 在你的站点上收集积极的赞扬并建立一个"共鸣"网页。 证人证言是一个推广你的应用的伟大的方式，对于广大的人群而言，来自第三方的赞美更值得信赖。

如果评论是负面的，你还要注意。显示你在认真听。 仔细地回应批评意见。可以这样说： "我们很感激你的反馈，但我们这样做是因为....." ,或者, "您提出一个好一点，我们正在加紧努力实现。"，这将软化对你的批评，并使你的产品显出了一副人性化面孔。在博客上的深思熟虑的评论可以驱散唱反调者，甚 至把抱怨的人转化成了你的福音传播者，这真是令人惊奇的事。

内线追加促销

在应用内部推销升级机会

谁都知道选择网站营销。但是销售不仅止于此。 如果你有一个分级的定价策略（或，你的应用有个免费版），不要忘了在产品内部大声呼吁升级的机会。

告诉人们 他们升级后，你会撤除限制。 例如： 在 Basecamp 中，使用免费帐号是不能上传文件的。 当有人试图上载一个文件，我们并不是简单的拒绝。 我们会解释，因何文件上传不能用，并鼓励他们升级到一个付费版本，并说明这样的好处。 同样的方法也用来鼓励现有用户当其现有规模已达极限时 升级到更高一级的用户帐户。

现存用户是你最佳销售对象。在向这些已经熟知并已使用你的产品的用户进行重复销售时，请不要害羞。

名字的魔力

给你的应用起个好记的名字

好多人都想给其应用起一个极其具有描述意义的 名字，这是一个大错误。不用担忧 挑一个能生动说明你的应用的用途的名字。 这通常会产生一个一般意义上的，容易被遗忘的名字。 Basecamp 是一个比什么 “项目管理中心” 或者 “项目特快”更好的名字。 Writeboard(写字板)这个名字要比 CollaborEdit(协作编辑)这个名字要好。

同样在起名过程中也别太指望群策群力。 挑选一个简短，上口，好记的名字并且马上使用之。

并且，如果你没有取得那个你想要的确切域名，也别急躁。 你经常可以有创造性的使用一些其它的字母。（例如： backpackit.com 或 campfirenow.com）

简化之

技术行业是否已经意识到了想出一个上口，自圆其说的名字会最终同样地受益？ 不管产品是啥，应该卖的更好，因为技术界将不会把 那些 认为自己被一群傲慢的工程师关到高科技俱乐部门外 的 消费 吓跑。 技术也应该迎头赶上。 新产品会更易描述，更易使用并且更易购买--- 这样，对公司而言， 意味着更易销售。

—David Pogue, 纽约时报 专栏作家 (摘自： 产品名字里有什么?)

感知痛苦

拆除研发和技术支持之间的墙壁

餐饮业中，在厨房干活和在前台服务顾客是 2 个截然不同的世界。这两边的相互理解和支持尤为重要。这就是烹饪学校和餐馆经常让厨师在前台作侍者服务顾客的原因，这样厨房的员工就可以和顾客接触并且体会到前台工作的真实情况。

很多软件开发人员也有类似的分工。 设计者和程序员在“厨房”工作， 支持人员应对客户。 不幸的是，这意味着软件大厨从来不会听到客户的真实意见。这是很有问题的，因为倾听客户的声音是你改善产品优缺点的最好途径。

解决方案？ 避免在你的客户和研发/设计之间竖起一堵墙。 **不要把客户支持外包给呼叫中心 或 第三方。你自己做。** 你，和你的整个团队，应该搞清楚客户的意见。 当客户烦恼时，你需要知晓。你要听他们的抱怨。你也要因此烦恼。

在 37signals 公司， 所有我们的技术支持邮件，都是产品的真正创建者亲自回复。 为何如此？ 首先， 这给客户更好的支持。 他们得到了 从构建应用的某人脑中想出的一个 直接回应。 其次，这使我们和使用我们的产品并且遇到问题的人们保持接触。 当他们受挫时，我们也受挫。 我们可以说， “我感知到了你的痛苦”，并且我们感同身受。

依靠统计分析来揭示问题点，很有诱惑力。但是统计数字和真实声音不可能一样。 你需要尽力去消除尽可能多的这种缓冲区，它挡在你和客户的真实声音之间。

前台是行为发生的地方。 去那里。 让你的大厨干侍者的活。 读客户邮件，听到他们的挫折，倾听他们的建议并且向他们学习。

拿掉中间人

在 "运作监控" 软件的开发所有时间里，支持和市场营销工作由 2 个人承担。 即便我们被迫扩充团队，我们也从不把技术支持从研发中分离出去。通过亲自对每个请求进行响应，我们强迫自己设身处地为客户着想，并且从他们的角度看问题。

理解你的客户因何需要某物很重要，不仅因为客户需要它。这种情形对于我们如何设计有直接影响。拿掉中间人。 当你把耳朵贴近地面仔细倾听时，会更容易地满足用户的需求。

我和很多人讨论过这种情形，他们的第一反应往往是： “你怎么不雇一个初级工程师来作技术支持？” ，为用户设身处地着想。 如果想要你的牛排按照你喜欢的方式烹饪，你是要和一个巴士男孩说，还是应该和那个真正在烹饪牛排的大厨讲呢？

***—David Greiner,** 运作监控* **创始人**

零培训

使用内嵌的帮助和常见疑难解答，产品就不需要手册或使用培训

你不需要一个手册去使用 Yahoo 或者 Google， Amazon。 因此， 为什么你不能也建立一个不需要手册的产品呢？ 力争建立一个这样的需要零培训的工具

你该怎样去做？ 好吧， 就像我们以前提到的，你开始就要保持简单。 你的应用越不复杂，你就越能免除帮助人们摆脱困境的麻烦。之后，一个伟大的事前支持途径是在潜在的容易引起疑惑的地方，使用内嵌的帮助和常见疑难解答。

例如, 我们在屏幕上给出事先支持，允许用户上传他们的 徽标。 有些人经历过这样一个问题，他们老是看到老的徽标，这是浏览器缓存引起的一个问题。 因此在"提交你的徽标"旁边的一个区域，我们增加了一个链接指向 这个常见疑难解答，来教客户 强行刷新 浏览器 以便看到新的徽标。 在我们这样做之前， 我们每天都要接到 5 个关于此问题的邮件。现在一个也没有了 。

快速回答

在疑难问题上的快速响应时间应该置于最高优先级

当你快速解答他们的问题时，客户很高兴。 他们已经习惯了录音电话式的支持，需要等几天（如果有的话），因此你可以通过提供体贴的即时响应 区别于你的竞争者。 在工作时间， 我们在 90 分钟内答复百分之九十的支持邮件，并且经常不到半个小时就完成了这项工作。人们喜欢这样。

即使你没有一个完美的答复，说点什么。 你可以用开放，诚实的方法 一个快速回复来赢得善意。 若有人抱怨有个不能马上得到解决的问题，象这样告诉他们, "我们已知道你所言那个问题，我们即将在不久之后仔细研究一下。" 这是驱散潜在负面困境的一个很好的方法

客户欣赏直率，并且经常转怒为喜，如果你用一种切中要害的方式快速响应。

混合部队

一个 3 人小团队怎样才能研发出一个创新型产品并且怎样成功地和那些大人物们竞争呢？答案就是招募一只混合部队

从你创业的第一天起，就要牢记客户是最重要的资产，他们对你的长远成功至关重要，因此对待社区用户要礼貌至上。和大人物竞争的方法是要从小处开始， 并且关注每个客户。

是你的客户第一个警告你，有 bug（程序缺陷），第一个警告你需求还没有得到满足。并且你的第一个客户会扛着你的大旗，替你广而告之。

这不是指你的产品要在上线时做到完美无缺。 恰恰相反，这是要你早早发布，常常发布。无论何时，当你的客户遇到 bug 时，一定要即时答复并且感谢他们的告知。

客户不指望你的产品完美无缺，也不指望你能实现他们所有想要的特性。然而， 客户一定想要你的倾听和你的关心，因此要显示你的关心。这是很多大公司表现尤为不足的地方，所以要早早建立一种社区归属感。

在 Blinklist, 每个客户的邮件都得到答复， 通常在第 1 个小时内（除非凑巧我们在睡觉）。 我们有个在线论坛，我们确保每个帖子和评论都会有回复。

同样重要的是， 所有我们的开发者都收到客户反馈，并且他们积极参与在线论坛。这样，我们慢慢地，确信地建立起了一个活跃的，忠实的 BlinkList 社区。

—*Michael Reining, MindValley & Blinklist* 的共同创始人

强硬的爱

乐于向客户说 不

至于特性需求，客户并不总是正确。如果我们把客户需求的每个零零碎碎都加入产品中，没有人会要我们的产品。

如果我们屈从一个客户的一时的兴致， Basecamp 就会有： 全面的时间跟踪， 全面的帐单，全面的会议安排，全面的日程安排，全面的附属任务系统，全面的即时消息聊天，全面的 wiki 功能，和全面的随便什么你能想到的。

然而， 我们通过客户调查得到的第一号 需求就是 要 保持 Basecamp 简单

这里还有另一个例子： 尽管有一些怨言， 我们决定产品不支持 IE5， 这样就有 7% 的市场被我们抹去了。但是我们人问我去关心剩下的 93% 更加重要。 为 IE5 修补产品错误和测试在时间上不划算。 我们宁愿为其它 93%的每个人作一个更好的产品。

作为一间软件研发公司，你必须扮演过滤器的角色。 并不是每个人建议的每件事都是正确答案。 我们认为客户所有的需求并不总是正确。 你不得不时常让一些人伤心失望。

关于这点， 作为一间研发公司热爱你的产品是最重要的。如果你的产品中掺和了一些你不认可的因素，你将不会爱你的产品。这是要否决你不以为然的客户需求是必须的，这一点的另一个明证。

In Fine Forum

Use forums or chat to let customers help each other

Forums and web-based group chat are a great way to let customers ask questions and help one another out. By eliminating the middleman — that's you — you provide an open stream of communication and save yourself time in the process.

At our product forums, customers post tips and tricks, feature requests, stories, and more. We pop in from time to time to offer some assistance but the forums are mainly a place for the community to help each other and share their experiences with the product.

You'll be surprised how much people want to help one another.

公开你的错误

拿出坏消息别让它挡道

如果某事出错就告诉人们。 即使他们开始并未曾发现。

例如， Basecamp 曾经在半夜宕机了数小时。 99%的顾客都未曾察觉，但是我们仍然在我们的 Basecamp 博客大全上张贴了“意外停机”的公告。 我们认为顾客该知道此事。

这是一个在我们出错时，张贴的公告的一个样本： “我们为今晨停机谨此致歉-我们有一些数据库问题，导致了某些用户的重大减速和故障。我们已经修复问题，并正在采取步骤，保证这个故障不会再次发生… …感谢你的耐心，并再次，为停机道歉。”

要尽可能开诚布公和透明。 不要保秘也不要遮遮掩掩。知情的客户是你最好的客户 。另外，你也会意识到大多的错误,在你的顾客的心中并不是那样地糟糕。 客户通常乐意给你一点喘息空间，只要他们知道你对他们是诚实的。

关于发布消息的旁注，好和坏： 当坏消息来时，立即把全部公开。另一方面 ，应该慢

慢地一点一滴地透露好消息，如果您能延长好的信息起到的效果，那么一定要这样作。

快速，直接和诚实

也许听起来奇怪，但是最佳情景是，当公司报告自身的坏消息时。 这是一个积极主动的举措，防止您的公司被置于不利的被动防御地位。

— 格雷戈 杉文，应用技术副总裁，CNET, 艾米丽 埃维拉, 校长, Calypso 通讯 (摘自： 危机公共入门)

一个月调整期

上线 30 天后发布一个重大更新

快速更新显示你的干劲。也表示你在听。还显示你有更多的后备招数。这使你能引起第二波的共鸣。加强了最初的好印象。 这给你一些谈资的和其他博客评论的话题。

明知快速升级迫近，使你在发布前把精力集中放在最关键的部件。 与其试图加入更多东西，不如开始真正完善核心功能群。 那么你就可以在真实世界推出产品。 一旦它在那里了，你可以开始收集客户的反馈意见，你就会知道其后的哪些方面需要注意。

这个婴儿学步的做法在 Backpack 行之有效。我们首先发布了基础产品，然后在数周后，加入一些新功能，象 Backpack 移动手持设备和标签，因为这些东西是我们客户告之的，他们最想要的功能。

保持发帖量

上线后维护一个持续的产品开发博客，显示你的产品充满活力

上线后不要停止写博客。 用一个专门的经常更新的博客，显示你的产品是充满活力的，（至少每周一次，如果可能应该时常更新） 。

这些事情包括：

* FAQ 疑难解答
* How-tos
* 小贴士 & 技巧
* 新功能，更新 & 补丁
* 共鸣/新闻

一 个博客不仅标志着你的应用充满活力，也使你的公司似乎更加人性化。 再次，不要害怕保持友善的和个性化的语气。小团队有时感觉需要保持一种听起来像大公司一样的语气，并且在所有时间里都要表现的及其专业。 这几乎就像一个商业版本的拿破仑综合症（译注：矮人自卑心理）。 不要因为听起来小就害怕。 事实上这其实很好，你可以跟客户像朋友那样聊天。

还活着

一个经常更新的产品博客是一个最好的网络应用正在积极发展的标志，，人们喜爱这个博客并且有人在家挑灯夜读。 被遗弃的产品博客是一个废弃产品的标志，人们会说，其负责人正趴在方向盘上睡觉

与用户在产品博客持续交流，要开诚布公并且慷慨的分享资讯。 让贵公司的哲学大放异彩。 公开链接并讨论竞争者。 暗示即将发布的功能，保持开放的意见反馈。

一个鲜活的产品，是一个与用户对话，并认真倾听的产品。 一个经常更新的产品博客提高了透明度，社区意识和对你的品牌的忠诚度。另外，免费宣传也是一种奖赏。

作 为Lifehacker 的编辑，我经常不断地浏览我所喜爱的产品的博客----例如：Google，flickr，Yahoo， el.icio.us 和37sigals 的产品博客。 比起那些单方面唐突发布版本，不与他们的用户和粉丝公开对话的公司，我更倾向于提起上述公司。
—Gina Trapani, Lifehacker,的编辑和Web 研发 生产力和软件指南

更好，而不是测试版

不要用"测试版"作替罪羊

这些日子感觉一切都是永远在测试阶段。 这真是一个好推辞。 一个无休止的测试阶段告诉客户你不是真的决心生产出成品。 像在说 ， "先用这个，但如果它不完美，这不是我们的错" 。

测试版将球推给你的客户。如果你对你的发行版没有足够的信心，你怎能期望公众会有呢？私下的测试版还好，公开的测试版简直就是胡扯。 如果产品对于公众使用而言还不足够好，就不要让公众去使用它。

不要等待你的产品尽善尽美。这不会发生。 对你发布的东西要负责。把它推出并称其为发布版。否则，你只是在找借口而已。

测试版是毫无意义

怪google 等公司造成了类似问题。 现在，用户已经被大量的研发人员训练出来了 ，

他们认为所谓 "测试版" 其实并不真正意味着什么。

--玛丽 侯德 ，信息架构师和交互设计师（摘自： Beta 测试的定义 ）

所有时间

仅仅是我，还是大家都在测试阶段，所有时间都在？

—- 吉姆 寇达 coudal 合伙企业 创始人

所有缺陷并不生而平等

分清缺陷的轻重缓急（甚至可以忽视其中一些）

只因为在你的产品中发现一个缺陷（bug） ，并不意味着这时候要引起恐慌。 所有软件都有缺陷---这是一个活生生的事实。

您不必每次马上就修补漏洞。 大多数缺陷（bug）是让人烦恼的，但都不是毁灭性的。 困扰问题可以搁置一会儿。 那些导致"看起来不太对"的错误的缺陷或其他小错误可以安全地预留一会儿。 如果一个缺陷（bug）破坏你的数据库，这时，显然需要修补。

分清缺陷的轻重缓急。 有多少人受到影响？ 问题坏到什么程度？ 这个缺陷 bug 值得立即重视吗或可以等待？ 你现在做什么将对绝大多数人产生最大影响？ 很多时候，加入新的功能，甚至比更改现有缺陷更为重要。

同时，要创造一种别害怕周围缺陷 bug 的文化。 缺陷发生，别总是找人去责怪。 你最不需要的就是这样一个环境，缺陷被私下解决，而不是通过公开讨论。

记得我们以前说过诚实的重要性。 如果客户抱怨的一个缺陷 bug ，可以与他们坦诚相见。 告诉他们你已经注意到这个问题并在处理。 如果不能马上纠正，你要说明理由，并且说明你在专注于改进产品的某些影响更广泛的方面。 诚实是最好的政策。

安度风暴

等到要求改变的应激反应停止后再采取行动

当你在小船上摇晃，将会激起波浪。 当你引入一个新的功能，改变了一个政策，或删除了什么，应激膝跳反应，往往负面的，就会涌入。

要抵制恐慌情绪，和拒绝作出迅速改变的反应。 激情在开始时闪耀。 但如果你安然度过

这最初的 24-48 小时，事情通常会平静下来。 大多数人在向你反应之前，他们并没有认真的使用和挖掘你添加的功能（或习惯你已经删除的那些功能） 。 所以你要坐稳，让这些反应都进来，并且在没有等待一段时间的情况下，别采取任何行动。 然后你可以采取一种更合理的反应。

还记得负面反应几乎总是高过正面的，而且更加充满激情。 事实上，你可能仅呢个听到负面声音，即使在大多数的用户对变化感到高兴的情况下。 请务必不要愚蠢到为了一个有争议的，但却必须要的决定，而作出后退的妥协。

保持先进性

订阅你的竞争对手新闻消息

订阅你的产品和你的竞争对手的新闻消息（知己知彼总是明智的）。 使用像 PubSub, Technorati, Feedster 的这些新闻反馈服务，得到最新更新（用关键字，公司名称和产品名称）。使用 RSS ，这个不断变化的信息源将把信息直接传递给你，是你总能跟上前进的速度。

小心那个臃肿的怪物

更成熟并不意味着更复杂

事物发展中，不要担心抵制臃肿。 诱惑将是做大。 但是那样并不总是正确。 只因为有些事物长大并且更加成熟， 但这不意味着要变得更加复杂

你并不需要成为一个外太空钢笔，要上下颠倒地书写。 有时成为一支铅笔就挺好。 你不需要是一把瑞士军刀。你可以作一把螺丝刀。 你不需要做一个潜水表，在 5000 米下还安全工作，假如你的客户是陆地爱好者，只想知道现在几点了。

不要因为膨胀而膨胀。这是许多应用变得臃肿的原因

新的并不总是意味着改进，有时你应该找准你的产品的定位点。

这是基于 Web 的应用优于传统桌面软件的主要好处之一。 桌面软件生产商 如：Adobe， Intuit，和微软 每年都要向你兜售新版本。 只因为他们不能总是卖给你相同的版本， 他们不得不通过添加新功能为收费提供正当理由。 从这里正式软件变得臃肿了

Web 软件是基于订阅收费的模型之上， 人们按月付费使用服务。 你不需要通过不断增加更多更多的功能来进行增值销售，你只需要提供一个持续的有价值的服务。

Go With the Flow

Be open to new paths and changes in direction

Part of the beauty of a web app is its fluidity. You don't wrap it up in a box, ship it, and then wait years for the next release. You can tweak and change as you go along. Be open to the fact that your original idea may not be your best one.

Look at Flickr. It began as a multiplayer online game called The Game Neverending. Its creators soon realized the photo-sharing aspect of the game was a more plausible product than the game itself (which was eventually shelved). Be willing to admit mistakes and change course.

Be a surfer. Watch the ocean. Figure out where the big waves are breaking and adjust accordingly.

Start Your Engines

Done!

Alright, you made it! Hopefully you're psyched to start Getting Real with your app. There really has never been a better time to make great software with minimal resources. With the right idea, passion, time, and skill, the sky's the limit.

A few closing thoughts:

Execution

Everyone can read a book. Everyone can come up with an idea. Everyone has a cousin that's a web designer. Everyone can write a blog. Everyone can hire someone to hack together some code.

The difference between you and everyone else will be how well you execute. Success is all about great execution.

For software, that means doing a lot of things right. You can't just have good writing but then fail to deliver on the promises in your prose. Clean interface design won't cut it if your code is full of hacks. A great app is worthless if poor promotion means no one ever knows about it. To score big, you have to combine all these elements.

The key is balance. If you tilt too far in one direction, you're headed for failure. Constantly seek out your weak links and focus on them until they're up to par.

People

It's worth reemphasizing the one thing that we think is the most important ingredient when it comes to building a successful web app: the people involved. Mantras, epicenter design, less software, and all these other wonderful ideas won't really matter if you don't have the right people on board to implement them.

You need people who are passionate about what they do. People who care about their craft — and actually think of it as a craft. People who take pride in their work, regardless of the monetary reward involved. People who sweat the details even if 95% of folks don't know the difference. People who want to build something great and won't settle for less. People who need people. OK, not really that last one but we couldn't resist throwing a little Streisand into the mix. Anyhow, when you find those people, hold onto them. In the end, the folks on your team will make or break your project — and your company.

More Than Just Software

It's also worth noting that the concept of Getting Real doesn't apply just to building a web app. Once you start grasping the ideas involved, you'll see them all over the place. Some examples:

- Special ops forces, like the Green Berets or Navy Seals, use small teams and rapid deployment to accomplish tasks that other units are too big or too slow to get done.
- The White Stripes embrace constraints by sticking to a simple formula: two people, streamlined songs, childlike drumming, keeping studio time to a minimum, etc.
- Apple's iPod differentiates itself from competitors by not offering features like a built-in fm radio or a voice recorder.
- Hurry up offenses in football pick up big chunks of yards by eliminating the "bureaucracy"

of huddles and play-calling.
- Rachael Ray bases her successful cookbooks and TV show on the concept of 30-minute "Get Real Meals."
- Ernest Hemingway and Raymond Carver used simple, clear language yet still delivered maximum impact.
- Shakespeare reveled in the limitations of sonnets, fourteen-line lyric poems in iambic pentameter.
- And on and on...

Sure, Getting Real is about building great software. But there's no reason why it needs to stop there. Take these ideas and try applying them to different aspects of your life. You might just stumble upon some neat results.

Keep In Touch

Let us know how Getting Real works out for you. Send an email to gettingreal [at] 37signals [dot] com.

Also, stay up to date with the latest offerings from 37signals by visiting Signal vs. Noise, our blog about Getting Real, usability, design, and a bunch of other stuff.

Thanks for reading and good luck!

37signals 资源

- 37signals 网站地址
- 信号 vs. 噪音 网络博客
- Basecamp — 基于 Web 的项目协调网络应用
- Campfire — 基于 Web 的商务群组聊天应用
- Backpack — 基于 Web 的信息组织工具
- Writeboard — 基于 Web 的协同写作工具
- Ta-da List — 基于 Web 的极简单的待办事宜列表工具
- Ruby on Rails — 开源 Web 应用框架