黑客与画家_Note

Preface

Paul Graham创业公式：

1. 搭建原型
2. 上线运营（别管bug）
3. 收集反馈
4. 调整产品
5. 成长壮大

- 只有明确用户需求，才能将失败的创业项目转变为成功的项目，前后业务方向可能会发生不断的改变，但是在公司发展方向不变的前提下，满足用户的需求，公司才能有进一步成长的机会。
- 黑客定义：好玩、高智商、探索精神，而非外行眼中的入侵者。
- 如果你想了解黑客，必须懂一点编程语言，就好比回到1880年，如果想了解技术发展，必须懂一点蒸汽机，放在现在也是适用的。
- 言论自由才能迸发思想的碰撞，独树一帜或者标新立异都能会让这个世界改变。

目录

什么是编程语言，为什么它们现在很热门

11. 一百年后的编程语言
    一百年后，人类怎样编程，为什么不从现在开始就这样编程呢
12. 拒绝平庸
    别忘了你的对手与你一样，能用任何想用的语言编写互联网软件
13. 书呆子的复仇
    在高科技行业，只有失败者采用"业界最佳实践"
14. 梦寐以求的编程语言
    一种好的编程语言，是让黑客可以随心所欲使用的语言
15. 设计与研究
    研究必须是"新的"，而设计必须是"好"的

正文

1. 为什么书呆子不受欢迎

智力本身与受欢迎无关，只是聪明的小孩子不想让自己在同龄人中受欢迎。
没有自信的人会通过贬低别人来抬高自己的地位。
聪明的小孩在读中学时往往是不快乐的，他们有其他的兴趣，没有多余精力让自己受欢迎，他们有自己思考着的东西，不会把所有的时间用来玩一种耗尽全力却又没有意义的游戏。
没有得到想要的工作，没有发挥自己的才能，就会导致一系列问题，青少年在技术没有达到一定水平的情况下，只能去做一些非技术相关的工作。
在这场激烈的人缘争斗中，书呆子并不是唯一的输家，只是因为他们安心干别的事去了，还有一些孩子则是主动放弃这些追逐，因为对这个过程感到厌恶。

2. 黑客与画家

黑客与画家、建筑师、作家一样，都是创造者，都在试图创造优秀的作品，但是在平常工作中，并没有特别明确的指标来衡量工作，更别提KPI，人们无法考核你的工作，可能就会被误解，最大才危险是连你自己也误解自己的 工作。
编程应该是解决实际的问题，而不是数学公式处理的问题。
眞正竞争软件设计的战场是新兴领域的市场，创业也只是从这里开始的，创业会面临两个问题，一是初期要包揽所有的活，二是赚钱的软件往往不是好玩的软件。
黑客如何才能做自己喜欢的事情，day job，night work，开源界的黑客们早就是这样做了，并且被其他领域的创造者验证过了，同时能够学习别人优秀的代码来提高自己。
画家学习绘画的方法主要是动手去画，黑客学习编程的方法也理应如此，并且定期回顾以往的编程项目，以此不断改进加入新的想法，使项目一直跟随自己的优秀。
承认规格设计的不完美，才能在编程的时候适当修改规格并考虑用户需求，最终得到一个更好的结果。
在恰当的时候与人合作，尝试换位思考，以一个不懂编程的角度出发，如何理解自己写的程序。
编程到底有多酷，取决于我们能够用编程做出怎样的工作。

3. 不能说的话

流行一时的不仅有衣服，还有道德观念，很多人会被习俗的力量所束缚，没有说出自己的想法。
有些正确的话，却不能说出口。
回顾过去，找那些过去被认为理所当然，如今被认为不可思议的事情，以此找出自己正在犯下的错误。

将当代观念与不同时期的古代观念diff一下，或参照不同的文化，将文化观念对比。
寻找那些一本正经的卫道者，看他们到底在捍卫什么。
观察禁忌如何产生。
流行的思想观念与流行的服饰产生方式不尽相同，但传播途径却很相似，第一批的接受者总是带有很强的抱负心，他们有自觉的精英意识，想把自己与普通人区分开来，当流行趋势确立后，第二批接受者就加入进来了，人数比上一批庞大得多，恐惧心在背后驱使着他们。
纯粹的好奇心，不想犯错，是找出不能说的话的理由。
智力越高的人，越愿意去思考那些惊世骇俗的思想观点。
训练自己去想那些不能想的事情，获得的好处会超过所得到的想法本身。
守口如瓶，笑脸相迎。
独处的力量远远超过你想象中的，若想能够清晰思考，必须寻找独处的机会。

4. 良好的坏习惯

大多数人把黑客当做破坏计算机的人，闻黑丧胆，但是，在Coding的世界中，黑客确是excellent的，唯有精通某一个领域，才有资格称得上黑客，从某种程度上来说，黑客首先是个优秀的程序员，其次才是黑客。
公民自由是国家富强的原因，而不是结果。

5. 另一条路

你的电脑向你的数据转换，数据从终端存储转换到中心存储，随时随地都可以获取操作数据，随之而来的是数据传输的效率和费用、数据传输及数据中心的安全性等问题。
软件协同办公。
找bug过程与安全测试类似。
软件架构设计与写作一样，新的构思会不断涌现。
人数越来越多，人与人之间需要的沟通呈现指数式增长，人越来越多，开会协同所需的时间越来越长，无法遇见的互相影响越来越大，产生的bug越来越多，反向也是成立的，人数越来越少，软件开发的效率将指数式增长。
“订阅报纸模式”是互联网软件天然的收费模式。
不管软件定价多少，有些用户永远都不会购买，盗版实际上是一种价格歧视，只不过针对的是最底层的消费者，很多公司都明白这个道理，只不过故意对某些盗版行为睁一只眼闭一只眼。
一般人都会懂得，把钱放在银行比放在家里安全，有些人就是不懂。
创业的两个原则：

1. 做出客户喜欢的产品，保证开支小于支出。
2. 如果你不打算自己动手设计和开发，那就不要创业。

6.如何创造财富

财富不等同于金钱。
致富最好的办法是自己创业或者加入创业公司，显然第二种做法在今年的某个时候被验证失败了。
创业公司往往与技术有关，本质上就是解决了某个技术难题的小公司。
程序员坐在电脑面前就能创造财富，一个优秀的程序员可以为公司创造很大的财富，相反，一个平庸的则可能无法创造财富甚至减少财富。
这个世界，你向下沉沦或向上奋进都取决于你自己，不能把原因推给外界。
你需要做一些人们需要的东西，即使不加入公司，你也能做到，真正重要的是做出人们需要的东西，而不是加入某个公司。
开发一个新的软件，会使你常规的维护和更新现有软件能创造更多的价值。
CEO是一种同时具备可测量性和可放大性的工作，一个表现糟糕的CEO是不能够推托说自己已经尽了全力的，如果公司表现不好，那就是他的表现不好。

可测量性：你的职位产生的业绩，应该是可测量的。（小团队合作）
可放大性：你做出的决定能够产生巨大的效应。（开发新技术）
小团队带来的各种额外激励会使每个成员的优势展现得淋漓尽致，因而小团队天生适合解决技术难题。
创业公司为每个人提供了一条途径，同时获得可测量性和可放大性。
选择公司要解决什么问题就应该以问题的难度作为指引，而且此后的各种决策都应该以此为原则。
创业的一些潜规则：

- 很多事情由不得你。
  真正创业后，你的竞争对手决定了你到底要有多辛苦，并且你能吃多少苦，对手也能吃多少苦。
- 创业的付出与回报虽然总体上是成比例的，但是在个体上不成比例。
  对于个人而言，付出与回报之间存在一个很随机的放大因子。

缓慢工作的后果并不仅仅是延迟了技术革新，而且很可能会扼杀技术革新，开发新技术是非常痛苦的经历，没有财富的激励，就不会有人去做技术革新。
只要懂得藏富于民，国家才会变得强大。

7. 关注贫富分化

当人们非常想把某件事做好的时候，有些人会做得比其他人好得多。
在现实世界中，财富是你必须自己创造出来的东西，而不是等着老爹买给你。
技术加剧了有技术者和无技术者之间的生产效率差异。

8. 防止垃圾邮件的一种方法

基于内容的过滤器，对单个词语进行贝叶斯判断，就能很好地过滤大部分垃圾邮件。

9. 设计者的品味

设计产品时，需要良好的品味。
好设计就是简单的设计。
好设计是永不过时的设计。如果你不愿意别人的答案取代你的答案，你就只好自己做出最佳答案。
好设计是解决问题的设计。
好设计是启发性的设计。
好设计通常是有点趣味性的设计。
好设计是艰苦的。
好设计是看似容易的。
好设计是对称的。
好设计是模仿大自然的。
好设计是一种再设计。
好设计是能够复制的设计。
好设计常常是奇特的设计。
好设计是成批出现的。
推动人才成批涌现的最大因素就是，让有天赋的人聚在一起，共同解决某个难题。
好设计常常是大胆的设计。
只有足够熟悉某个领域，才可能发现哪些地方可以改进。

10. 编程语言解析

程序员的时间比计算机的时间昂贵得多。

长期使用某种语言，就会慢慢按照这种语言的思维模式进行思考。
静态类型语言与动态类型语言没有绝对的好坏之分，每个人都可以有自己的选择。

11. 一百年后的编程语言

语言只是一种书写法，程序则是一种严格符合规则的描述，以书面形式记录计算机如何解决你的问题。
随着技术的发展，每一代人都在做上一代人觉得很浪费的事情。
浪费分成好的浪费和坏的浪费，感兴趣的就是好的浪费，即用更多的钱得到更简单的设计。
对速度的追求是人类内心根深蒂固的欲望，设计编程语言的时候，应该有意识地问问自己，什么时候可以放弃一些性能，换来一点点便利性的提高。
将语言的语义与语言的实现分离。
写文章，就是为了试着搞清楚某件事情，对于软件而言也是如此。
效率低下的软件并不等于很烂的软件，一种让程序员做无用功的语言才真正称得上很烂，浪费程序员的时间而不是浪费机器的时间才是真正的无效率。
面向对象编程，使得可以对代码进行可持续性开发。
性能分析器有助于指导程序开发，现在许多人仍然相信程序运行速度提升的关键在于开发出能够生成更快速代码的编译器。

- 一百年后的编程语言在理论上今天就能设计出来。
- 如果今天真的能设计出这样一种语言，很可能现在适合编程，并且能够产生更好的结果。

12. 拒绝平庸

雷蒙德写过一篇文章《如何成为一个黑客》提到，如果想成为一个黑客，建议从Python和Java入手，因为它们比较容易学，想当高级一点的黑客，还应该学习C和Perl，前者用来对付Unix系统，后者用来系统管理和开发CGI脚本，最后，真正非常严肃地把黑客作为人生目标的人，应该考虑学习Lisp：

> Lisp很值得学习，你掌握它以后，会感到它给你带来的极大启发，这会大大提高你的编程水平，使你成为一个更好的程序员，尽管在实际工作中极少会用到Lisp。

选择哪种技术的时候，只能考虑选择什么样的技术能最好地完成工作。
如果一个创业公司每年成长10%，那就意味着要关门倒闭了。
选择一门能够快速开发的语言，会让公司一直具有活力。
编程语言不仅仅是技术，也是一种习惯思维，非常难于改变。
唯一洞悉所有语言劣势的人必然是懂得最强大的那种语言的人。

13. 书呆子的复仇

在某些情况下，一些语言就是比另一些语言更出色。
认为所有语言都一样的看法的缺点是自欺欺人，但优点是可以使许多事情变得很简单。
编程语言现在的发展不过刚刚赶上1958年Lisp语言的水平。
Lisp刚诞生的时候包含的9种思想：

1. 条件结构(即if-else结构)
2. 函数也是一种数据结构
3. 递归
4. 变量的动态类型
5. 垃圾回收机制

6. 程序由表达式组成
7. 符号类型
8. 代码使用符号和常量组成的树形表示法
9. 无论什么时候，整个语言都是可用的（Lisp不真正区分读取期、编译期、运行期）

随着时间流逝，流程的编程语言不断更新换代，语言设计思想逐渐向Lisp靠拢，思想1-5已经被广泛接受，思想6开始在主流编程语言中出现，思想7在Python中有所实现，不过似乎没有专用的语法。
思想8和9原本不属于lisp语言的原始构想，但这也成了Lisp独一无二的特点。意味着你可以写出一种能够自己编程的程序。
条件越苛刻的项目，强大的编程语言就越来越能发挥作用。
使用不常见的语言出现的问题可能有三个：

1. 写出的程序可能无法很好地与使用其他语言写的程序协同工作。
2. 可能找不到很多函数库。
3. 可能不容易雇到程序员。

如果你创业的话，千万不要为了取悦风险投资商或潜在并购方而设计你的产品，让用户感到满意才是你的设计方向，只要赢得用户，其他事情就会接踵而来。
语言的编程能力越大，写出来的程序就越短，高级语言是能够提供更强大抽象能力的语言，从某种意义上，就像提供更大的砖头，砌墙的时候用到砖头数量就少了。
技术本该极端。
两个结论：

1. 不同语言的编程能力不一样。
2. 大多数经理故意忽视第一点。

使用最强大的语言，解决最难的问题。

14. 梦寐以求的编程语言

大多数人选择某一种编程语言，不是因为这种语言有什么独特的特点，而是因为听说其他人使用这种语言。
对于什么是优秀编程语言，黑客的看法与大多数的语言设计者不一样，编程语言不是数学定理，而是一种工具，专家级黑客一眼就能认出语言的好坏。
流行程度也会使语言更加优秀。
一种语言必须是某一个流行计算机操作系统的脚本语言。
编程语言，一种免费的实现，一本相关书籍，语言所依附的计算机系统，简洁性，可编程性。
一种真正优秀的编程语言应该既整洁又混乱，整洁，设计得很清楚，内核由数量不多的运算符构成，这些运算符易于理解，每一个都有很完整的独立用途，混乱，允许黑客以自己的方式使用。
开发程序应当先从简单的功能入手，即一次性程序，然后不断改进。
新技术被市场接纳的方式有两种，一种是自然成长式，另一种是大爆炸式。
为了编写优秀软件，必须同时具有两种互相冲突的信念，一方面，要像初生牛犊一样，对自己的能力信心万丈，另一方面，又要像历经沧桑的老人一样，对自己的能力抱着怀疑态度。
实际上，二者并不矛盾，乐观主义和怀疑倾向分别针对两个不同的对象，必须对解决难题的可能性保持乐观，同时对当前解法的合理性保持怀疑。
做出优秀成果的人，在做的过程中常常觉得自己做得不够好，在其他人看来已经是够好了，而创作者本人看到的都是自己作品的缺陷。
一开始的时候要精心选择用户，避免使用者过快增长，发展用户就像一种优化过程。

15. 设计与研究

让用户满意并不等于迎合用户的一切要求，用户不了解所有可能的选择，也经常弄错自己真正想要的东西。
编程语言也应该是以人为本。
贴近用户的设计思想被归纳为“弱即是强”模式。
任何时候，代码都必须能够运行。