**《黑客与画家》摘抄**

前言

软件带来财富，仅仅代表了大趋势的一面而已。这种大趋势就是本书的主题。我们的时代是计算机时代。以前，人们曾经认定这个时代应该是太空时代或者原子时代。但是事实证明，它们只是公关公司发明的概念。计算机对人类生活的影响远远超过了太空航行或者原子技术的影响。

计算机程序只是文本而已。你选择什么语言，决定了你能说什么话。编程语言就是程序员的思维方式。

为什么黑客那么在乎言论自由？我认为，部分原因在于，革新对于软件行业实在是太重要了，而革新和异端实际上是同一件事。优秀的黑客养成了一种质疑一切的习惯。这是肯定的，因为如果你不得不同一台机器打交道，而这台机器全部由文字组成，像机械式手表一样复杂，并且规模大出1000倍，那么你也会养成这种习惯的。

1 为什么书呆子不受欢迎（他们的心思在别的地方）

所以，如果智力本身与“受欢迎”无关，为什么聪明的小孩一直不受同龄人的欢迎呢？我认为，答案就是他们真的不想让自己受欢迎。

虽然“书呆子”饱尝不受欢迎之苦，但是为了解除痛苦而让他们放弃“聪明”，我想大多数人是不会愿意的。对他们来说，平庸的智力是不可忍受的。我认为，这就是问题的根源。“书呆子”的目标具有两重性。他们毫无疑问想让自己受欢迎，但是他们更愿意让自己聪明。

同伴的意见成为他们判别事物的标准，这不仅体现在衣着上，还体现在他们做的几乎每一件事情上，就连走路的姿势也不例外。所以，他们为了把所有事情“做对”，所付出的任何努力，不管是有意还是无意，实际上都等同于努力在使自己变得更受欢迎。

书呆子不受欢迎的真正原因，是他们脑子里想着别的事情。他们的注意力都放在读书或者观察世界上面，而不是放在穿衣打扮、开晚会上面。

在一个人产生良知之前，折磨就是一种娱乐。

难怪聪明的小孩读中学时往往是不快乐的。他们有其他的兴趣，没有多余的精力用来使自己更受欢迎。你在其他地方有所得，就会在这个地方有所失。不受欢迎使得书呆子成为全校攻击的目标。令人惊奇的是，这种噩梦般的情景并非出自任何有预谋的恶意，而仅仅因为这个特殊的环境。

没错，成年人不知道孩子们内部发生的事。认识到这一点很重要。在抽象意义上，成年人知道孩子的行为有时是极端残酷的，这正如我们在抽象意义上知道贫穷国家的人民生活极端艰难。但是，像所有人一样，成年人不喜欢揪住不放这种令人不快的事实。你不去埋头探寻，就不会发现具体的证据，就会永远以为这件事是抽象的。

我认为，真实世界的关键并非在于它是由成年人组成的，而在于它的庞大规模使得你做的每件事都能产生真正意义上的效果。

真实世界的特点是，它极其庞大。如果总体足够大，即使是人数最少的少数派，只要聚集在一起，也能产生可观的力量。

如今的成年人根本不接受青少年。一般来说，他们都是在办公室工作，所以就在上班的路上，顺路把孩子送到学校去关着，这有点像他们周末外出度假时，把狗送到寄养的地方。

几乎在任何团体中都存在等级关系。成年人在真实世界中形成的团体，一般来说，都存在某个共同目标，团体的领导者往往由最善于实现目标的人担任。学校就不一样，大多数情况下，学生内部形成的团体没有一个共同目标。但是，等级关系却不会缺席，所以孩子们的等级是凭空创造出来的。

校园生活的真正问题是空虚。除非成年人意识到这一点，否则无法解决这个问题。

2 黑客与画家（黑客也是创造者，与画家、建筑师、作家一样）

黑客与画家的共同之处，在于他们都是创作者。与作曲家、建筑师、作家一样，黑客和画家都是试图创作出优秀的作品。他们本质上都不是在做研究，虽然在创作过程中，他们可能会发现一些新技术。

与其说优秀的软件设计师是工程师，还不如说是建筑师。建筑学和工程学之间的区别并不是很严格的，但就是存在区别。这表现在“做什么”和“怎么做”：建筑师决定做什么，工程师想出怎么做。 ？？？

创造优美事物的方式往往不是从头做起，而是在现有成果的基础上做一些小小的调整，或者将已有的观点用比较新的方式组合起来。这种类型的工作很难用研究性的论文表达。

唯一有效的外部考核就是时间。经过岁月的洗礼，优美的东西生存发展的机会更大，丑陋的东西往往会被淘汰。不幸的是，这种考核需要的时间可能比一个人的生命还要长。

※※在读研究生期间，我潜意识里一直有一种很不舒服的感觉，觉得自己应该多学一点理论，不应该期末考试结束还不到三个星期，就把所有东西忘得一干二净，那样真是不可饶恕。现在，我意识到自己错了。黑客搞懂“计算理论”的必要性，与画家搞懂颜料化学成分的必要性差不多大。我发现，黑客新想法的最佳来源，并非那些名字里有“计算机”三个字的理论领域，而是来自于其他创作领域。

我现在认为，大学里交给我的编程方法都是错的。你把整个程序想清楚的时间点，应该是在编写代码的同时，而不是在编写代码之前，这与作家、画家和建筑师的做法完全一样。明白这一点对软件设计有重大影响。它意味着，编程语言首要的特性应该是允许动态扩展。编程语言是用来帮助思考程序的，而不是用来表达你已经想好的程序。

真正竞争软件设计的战场是新兴领域的市场，这里还没有人建立过防御工事。只要你能做出大胆的设计，由一个人或一批人同时负责设计和实现产品，你就能在这里战胜大公司。

价格是由供给和需求共同决定的。好玩的软件的需求量，比不上解决客户麻烦问题的软件的需求量。在小剧场里演出的酬劳，比不上穿着卡通大猩猩服装、在展览会上为厂商站台的酬劳。写小说的回报比不上写广告文案的回报。开发编程语言的收入，比不上把某些公司老掉牙的数据库连上服务器的收入。

因为如果你不爱一件事，你不可能把它做得真正优秀，要是你很热爱编程，你就不可避免地会开发你自己的项目。

同样地，黑客可以通过观看优秀的程序学会编程，不是看它们的执行结果，而是看它们的源代码。开源运动最鲜为人知的优点之一，就是使得学习编程变得更容易了。

你不能盼望先有一个完美的规格设计，然后再动手编程，这样想是不现实的。如果你预先承认规格设计是不完美的，在编程的时候，就可以根据需要当场修改规格，最终会有一个更好的结果。

优秀的软件也要求对美的狂热追求。如果你查看优秀软件的内部，就会发现那些预料中没有人会看见的部分也是优美的。

如果黑客只是一个负责实现领导意志的技术工人，职责就是根据规格说明书写出代码，那么他其实与一个挖水沟的工人是一样的，从这头挖到那头，仅此而已。

如果一个代码块由三四个人共同开发，就没有人真正“拥有”这块代码。最终，它就会变得像一个公用杂物间，没人管理，又脏又乱，到处堆满了冗余代码。

事实表明，从他人的角度思考问题正是成功的奥秘所在。“换位思考”并不就意味着你要做自我牺牲。实际上，这是完全不同的两回事。了解别人对于事物的看法，并不代表你为他的利益服务。

《计算机程序的结构与解释》卷首语：程序写出来是给人看的，附带能在机器上运行。

在达·芬奇的年代，绘画并不是一件很酷的事情，达·芬奇用自己的工作推动绘画成为一种伟大的表达方式。同样，编程到底能够有多酷，取决于我们能够用这种新媒介做出怎样的工作。

3 不能说的话（如果你的想法是社会无法容忍的，你怎么办？）

历史的常态似乎就是，任何一个年代的人们，都会对一些荒谬的东西深信不疑。他们的信念还很坚定，只要有人稍微表示一点怀疑，就会惹来大麻烦。

我们这个时代是否有所不同？只要读过一点历史，你就知道答案几乎确定无疑，就是“没有不同”。即使有那么一丝微小的可能，有史以来第一次，我们这个时代的所有信念都是正确的，那也是出于惊人的巧合，而不是因为我们真找到了正确的方向。

你是一个随大流的人吗？

与历史上别的年代一样，我们的思想几乎肯定也是一张有错误的地图。如果你也犯下与别人一样的错误，那么这个错误不太可能完全来自于你自己。

真话

我猜想，最令人暴跳如雷的言论，就是被认为说出了真相的言论。

异端邪说

历史上的每一个年代，都会给“异端邪说”贴上一些标签，目的是在人们开始思考它们是否为真之前就把它们封杀。

我们可以自以为是地相信，当代人比古人更聪明、更高尚。但是，了解的历史越多，就越明白事实并非如此。古人与我们是一样的人，他们既不是更勇敢，也不是更野蛮，而是像我们一样通情达理的普通人。不管他们产生怎样的想法，都是正常人产生的想法。

不管实际上是否有思想斗争，斗争总是会以思想斗争的形式表现出来。正如刚刚过气的时尚并非一点儿也不时尚，失败一方的思想其实也并非一无是处。

优秀作品往往来自于其他人忽视的想法，而最被忽视的想法就是那些被禁止的思想观点。

做一个异端是有回报的，不仅是在科学领域，在任何有竞争的地方，只要你能看到别人看不到或不敢看的东西，你就有很大优势。

“守口如瓶”的真正缺点在于，你从此无法享受讨论带来的好处了。讨论一个观点会产生更多的观点，不讨论就什么观点也没有。所以，如果可能的话，你最好找一些信得过的知己，只与他们畅所欲言、无所不谈。这样不仅可以获得新观点，还可以用来选择朋友。

如果你的思想很保守，你自己不会知道，而且你很可能还会持有相反的看法。请记住，所谓“流行”（传统观念也是一种流行），本质上就是自己看不见自己的样子。否则就不会有流行了。

时间就是一种产生距离的简单方法，实际上，新的流行让旧的流行更容易被观察到，因为对比之下，旧的流行会显得很荒唐。从钟摆波动的一端望去，上一个周期的端点就显得特别遥远。

小时候，每个人都会鼓励你不断成长，变成一个心智成熟、不再耍小孩子脾气的人。但是，很少有人鼓励你继续成长，变成一个怀疑和抵制社会错误潮流的人。

如果自己就是潮水的一部分，怎么能看见潮流的方向呢？你只能永远保持质疑。问自己，什么话是我不能说的？为什么？

4 良好的坏习惯（与其他美国人一样，黑客的成功秘诀就是打破常规）

只有深入了解当前的技术，黑客才能构想下一代技术。知识产权的拥有者也许会说，不，谢谢，我们不需要你的帮助，我们自己就能开发下一代技术。他们错了，在计算机工业的历史上，新技术往往是由外部人员开发的，而且所占的比例可能要高于内部人员。

黑客的这种自行其是的特点，很大程度上说明了，为什么不管是出色的工作还是糟透了的工作，黑客都用同一个词（terrific）形容。如果他们做出来一个东西，他们自己总是无法百分百确定那到底是什么东西，有可能完全没用。但是只要那些出错的地方还算正常，那么就是一个信号，表明这个东西还有希望。

这就是为什么黑客感到担忧。政府侵犯公民自由，从表面上看，并不会让程序员的代码质量下降。它只是逐渐地导致一个错误观点占上风的世界。黑客对于公民自由是非常敏感的，因为这对他们至关重要。他们远远地就能感到极权主义的威胁，好比动物能够感知即将来临的暴风雨。

那些占据高位、本能地想要约束黑客、强迫黑客服从的人们，请小心你们的要求，因为你们真有可能成为千古罪人。

很自负的人必须培养出敏锐的感觉，及时发现周围情势的变化，知道怎样才能脱身。

5 另一条路（互联网软件是微机诞生后的最大机会）

大多数人，在大多数时候，总是选择最省事的做法。

“你的电脑”这个概念正慢慢成为过去时，取而代之的是“你的数据”。你应该可以从任何电脑上获取你的数据。或者更准确地说，在任何终端设备上获取你的数据，终端设备不一定是电脑。

互联网应用程序能够同时被多人使用，所以非常适合团队协作性的工作。

对于用户来说，使用互联网软件不会有太大的压力。我觉得，大多数Windows用户使用桌面软件的时候都会感到紧张，会有相当大的心理压力。施放这种压力，对你的产品将是一种巨大的推动。

但是，硬件需要考虑的地方，不仅仅在于怎么才能避免出问题，还在于怎样才能最大地发挥它们的作用。

对于互联网软件，没人规定只能使用某些语言开发，因为所有的硬件都控制在你手里，你想要用什么语言，就能用什么语言。不同的语言适合不同的人物，你应该根据不同的场合，挑选最合适的工具。尤其是在竞争者存在的情况下，“可以这样做”就变成了“必须这样做”，因为如果你不利用语言的优势，那就会听任对手超过你。

互联网软件的发布规则是：它运行不了，你就无法发布；一旦它能运行了，你就可以立刻发布。

不要被微软吓到。你能做到它做不到的事情，正如它能做到你做不到的事情一样。开发互联网软件不需要得到任何人的许可，没有任何能够阻止你。你不需要去申请许可证，不需要在零售店的货架上谋得一席之地，也不需要卑躬屈膝地求人家，将你的软件与操作系统捆绑在一起。你能够通过浏览器发布软件，没有人能在你和浏览网站的用户之间插上一脚。

6 如何创造财富（致富的最好方法就是为社会创造财富。创造财富的最好方法就是创业）

从经济学观点看，你可以把创业想象成一个圧缩过程，你的所有工作年份被压缩成了短短几年。你不再是低强度地工作四十年，而是以极限强度工作四年。在高技术领域，这种压缩的回报尤其丰厚，工作效率越高，额外报仇就越高。

通过创造有价值的东西而致富，这种方法的优势不仅仅在于它是合法的，还在于它更简单。你只需要做出别人需要的东西就可以了。

财富才是你的目标，金钱不是。但是，如果财富真的这么重要，为什么大家都把挣钱挂在嘴边呢？部分原因是，金钱是财富的一种简便的表达方式；金钱有点像流动的财富，两者往往可以相互转化，但是，他们确实不是同样的东西，除非你打算伪造货币。

我们这个世界，你向下沉沦或者向上奋进都取决于你自己，不能把原因推给外界。许许多多不创造任何财富的人——比如本科生、记者和政客——一听到最富有的5%人口占有全社会一半以上的财富，往往会认定这是不公平的。一个有经验的程序员很可能也认为这是不公平的。因为最顶尖的5%的程序员写出了全世界99%的优秀软件。

公司的一切行为的目的都是盈利，从而生存下去。创造财富是大多数公司盈利的手段。公司的业务高度专业化，掩盖了它们都是在创造财富的这种相似性，你不要觉得只有制造业公司在创造财富。

公司不过是一群人在一起工作，共同做出某种人们需要的东西。真正重要的是做出人们需要的东西，而不是加入某个公司。对于大多数人来说，最好的选择可能是为某个现存的公司打工。但是，理解这种行为的真正含义对你没有什么坏处。工作就是在一个组织中，与许多人共同合作，做出某种人们需要的东西。

要致富，你需要两样东西：可测量性和可放大性。你的职位产生业绩，应该是可测量的，否则你做得再多，也不会得到更多的报酬。此外，你还必须有可放大性，也就是说你做出的决定能够产生巨大的效应。

理想情况下，你与其他愿意更努力工作的人一起组成一个团队，共同谋取更高的回报。因为创业公司的团队往往是自发形成的，许多有抱负的创始人彼此之间早就相识（至少听说过对方），所以他们对彼此贡献的评估要比一般的小团体更准确。创业公司不仅仅是十个人的团队，而且是十个同类人的团队。

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。因为创业公司是小团队，所以具备可测量性。因为创业公司通过发明新技术盈利，所以具备可放大性。

小团队天生就适合解决技术难题。技术的发展是非常快的，今天很有价值的技术，几年后可能就会丧失价值。小团队在如今这个时代可谓如鱼得水，因为他们不受官僚主义和繁琐管理制度的拖累。而且，技术的突破往往来自非常规的方法，小团队就较少受到常规方法的约束。

创业是有一些潜规则的，其中一条就是很多事情由不得你。比如，你无法决定到底付出多少。另一条潜规则是，创业的付出与回报虽然总体上是成比例的，但是在个体上是不成比例的。

创业公司不像能经受打击的黑熊，也不像有盔甲保护的螃蟹，而是像蚊子一样，不带有任何防御，就是为了达到一个目的而活着。蚊子唯一的防御就是，作为一个物种，它们的数量极多，但是作为个体，却极难生存。

用户数量也许不是最好的测量指标，但应该也相差不远了。买家关心它，收入依赖它，竞争对手恐惧它，记者和潜在用户则是被它打动。无论你的技术水平有多高，用户数量都比你自己的判断更能准确反映哪些问题应该优先解决。

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。只有在快速获得巨大利益的激励下，你才会去挑战那些困难的问题，否则你根本不愿意去碰它们。开发新技术是非常痛苦的经历，正如爱迪生所说，百分之一的灵感加上百分之九十九的汗水。没有财富的激励，就不会有人愿意去做技术革新。工程师愿意接受普通薪水去做一些诱人的项目，而与日常生活关系更密切的技术革新只能由创业者发明。

一旦自己的财产有了保证，那些想致富的人就会愿意去创造财富，而不是偷窃。

7 关注贫富分化（“收入分配不平等”的危害，会不会没有完美想的那样严重？）

那些专门把运动员和CEO的收入拿出来说事的社论让我想到了中世纪的基督教教士。他们宁愿高谈阔论地球是不是圆的，也不愿亲自研究一下现实中的证据。一个人的工作具有多少价值不是由政府决定的，而是由市场决定的。

因为要是你现在还处于“老爹”模式，认为财富就是从某个口子流出来、被大家分享的东西，而不是来源于满足他人的需求的创造活动，那么当你注意到有些人赚钱比其他人多得多时，你就会不偏不倚地得出“不公平”这个结论。

技术对生产效率的提高不是线性的，而可能是多项式形态的。所以，随着时间推移，我们应该会看到个人生产效率总是保持增长。这种增长会使得贫富差距不断扩大吗？这取决于你指的是什么“差距”。

技术无法使其变得更便宜的唯一东西，就是品牌。这正是为什么我们现在越来越多地听到品牌的原因。富人与穷人之间生活差异的鸿沟正在缩小，品牌是这种差距的遗留物。

在一个剥夺个人财产的社会，财富创造活动中所有那些没有乐趣的事情都会急剧地放慢，乃至停顿。

一个社会需要有富人，这主要不是因为你需要富人的支出创造就业机会，而是因为他们在致富过程做出的事情。

8 防止垃圾邮件的一种方法（不久前，许多专家还认为无法有效地过滤垃圾邮件。本文改变了他们的想法）

开发垃圾邮件过滤器时，统计学方法往往不是程序员首先想到的方法。大多数黑客的直觉是写出一个能够识别垃圾邮件某种特征的软件。

贝叶斯方法的真正优点在于你知道你正在计算的是什么东西。

我以前曾经认为，那些相信更严格的法律会遏制垃圾邮件的人真是太天真了。我现在认为，更严格的法律或许无法减少我们收到的垃圾邮件的数量，但是肯定有助于减少逃过过滤器拦截的垃圾邮件的数量。

不过，虽然接近于零，发送垃圾邮件还是有成本的。所以，只要我们把垃圾邮件的回应率降得很低，商家就会发现，发送垃圾邮件是一件经济上不值得的事情

9 设计者的品味（如何做出优秀的东西？）

喜欢一件东西，却不知道为什么自己喜欢它，原因可能是这件东西是美的，但也可能因为他们的母亲也拥有同样的东西，或者杂志上某个明星使用它，或者仅仅因为它的价格很昂贵。人类的思想就是没有经过整理的无数杂念的混合。

大人教导你说品味只是每个人的偏好而已。但是来到博物馆，他们却对你说，仔细观赏达·芬奇的作品，因为他是伟大的艺术家，品味超凡。

把品味说成个人的偏好可以有效地杜绝争论，防止人们争执哪一种品味更好。但是问题是，这种说法是不正确的。只要你自己开始动手设计东西，就能明白这一点。

就像别的工作一样，只要你不断地从事设计工作，你就会做得越来越好。你的品味会出现变化，你会像别人一样有所提高。如果这样的话，那么你以前的品味就不只是与现在不同，而是不如现在的好。因此 ，所谓的“品味没有好坏之分”的公理也就顿时见鬼去了。

只要你走出狭隘的自我，至少在心理对自己说，确实存在比其他设计更好的杰出设计，那么你就能开始仔细研究了。你的品味是如何变化的？什么原因使你做出不好的设计？其他人对设计是什么观点？

当你被迫把东西做得很简单时，你就被迫直接面对真正的问题。当你不能用表面的装饰交差时，你就不得不做好真正的本质部分。

如果解决方法是丑陋的，那就肯定还有更好的解决方法，只是还没有发现而已。

以永不过时作为目标也是一种避开时代风潮的影响的方法。“风潮”这个词，从字面上就可以看出，它就是一阵风似的，随着时间经常改变。如果一件东西长盛不衰，那么它的吸引力一定来自本身的魅力，而不是来自风潮的影响。

答案可以不断改进，同样，问题本身也可以不断改进。软件的难题通常可以被改成等价的较易解决的形式。历史上，物理学的主要难题曾经一度是如何诠释经典著作，后来逐渐变成对可观测到的行为进行预测，这种转变使得物理学的发展速度大大加快。

好设计是启发性的设计，在软件业中，这条原则意味着，你应该为用户提供一些基本模块，使得他们可以随心所欲自由组合，就像玩乐高积木那样。

困难的问题需要艰巨的付出才能解决，高难度的数学证明需要结构非常精细的解决方法（它们往往做起来很有趣），工程学也是如此。

当你攀登高山时，必须扔掉一切不必要的装备。在困难地点或预算不足的条件下，建筑师就只能做出很简练的设计。当解决难题成为压倒一切的任务时，那些流行样式与华丽装饰就被抛到一边去了。

并非所有的痛苦都是有益的。世界上有有益的痛苦，也有无益的痛苦。你需要的是咬牙向前冲刺的痛苦，而不是脚被钉子扎破的痛苦。解决难题的痛苦对设计师有好处，但是对付挑剔的客户的痛苦或者对付质量低劣的建材的痛苦就是另外一回事了。

练习的作用也许是训练你把刻意为之的事情变成一种自觉的行为。有时，我们的训练只是为了让身体养成下意识的反应。

在数学和工程学中，递归尤其有用。归纳式证明方法既简洁又美妙。在软件中，能用递归解决的问题通常代表已经找到了最佳解法。

对称的危险在于它可以用来取代思考，在大量使用重复的时候这种危险性更大。

犯错误是很正常的事情。你不要把犯错看成灾难，要勇于承认、勇于改正。

我们对待复制的态度经常是一个否定之否定的过程。刚入门的新手不知不觉地模仿他人，逐渐熟练之后才开始创作原创性作品。最后他会意识到，把事情做对比原创更重要。

你最后发展出来的风格是自然而然形成的。“奇特”这个特点尤其如此，没有其他路可走。它就像连接大西洋和太平洋的“西北航道”，无数人希望找到这条捷径。16世纪的风格主义者、19世纪的浪漫主义者、一代代的美国高中生都在寻找，但就是找不到。唯一达到“奇特”的方法，就是追求做出好作品，完成之后再回过头看。

大多数做出优美成果的人好像只是为了修正他们眼中丑陋的东西。伟大成果的出现常常来源于某人看到一样东西后，心想我能做得比这更好。

10 编程语言解析（什么是编程语言？为什么它们现在很热门？）

编程语言的一个重要特点：一个操作所需的代码越多，就越难避免bug，也越难发现它们。

那么，应该使用哪一种语言？嗯，关于这个问题，现在有很多争论。部分原因是，如果你长期使用某种语言，你就会慢慢按照这种语言的思维模式进行思考。所以，后来当你遇到其他任何一种有重大差异的语言，即使那种语言本身并没有任何不对的地方，你也会觉得它极其难用。缺乏经验的程序员对于各种语言优缺点的判断经常被这种心态误导。

有意思的是，劫持飞机与“缓冲区溢出攻击”有类似之处。在一般飞机上，乘客区与驾驶舱是想通的，就好像C语言中数据区与代码区是相邻的一样。劫机者一旦进入驾驶舱，实际上就相当于把自己从数据提升为代码。

关于面向对象编程优劣的争论并不像静态类型与动态类型之争那样壁垒分明，因为编程的时候你只能在静态类型和动态类型中选一种。但是，面向对象编程只是程度不同的问题。事实上有两种程度的面向对象编程：某些语言允许你以这种风格编程，另一些语言则强迫你一定要这样编程。

程序员活在这个文艺复兴时代可能是一件好事。如果我们所有人都使用同一种编程语言，反而有可能是坏事。

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？）

编程语言就像生物物种一样，存在一个进化的脉络，许许多多分支最终都会成为进化的死胡同。这种现象已经发生了。Cobol语言曾经流行一时，但是现在看来没有任何后续语言继承它的思想。它就像尼安德特人一样，进化之路已经走到了尽头。

编程语言之所以可能出现聚合，一个原因是它的概率空间比较小，另一个原因是它的突变不是随机的。语言的设计者们总是有意识地借鉴其他语言的设计思想。

编程语言进化缓慢的原因在于它们并不是真正的技术。语言只是一种书写发，而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应该如何解决你的问题。所以，编程语言的进化速度更像数学符号的进化速度，而不像真正的技术的进化速度。数学符号的进化是缓慢的渐变式变化，而不是真正技术的那种跳跃式发展。

对速度的追求是人类内心深处根深蒂固的欲望。当你看着计算机这个小玩意，就会不由自主地希望程序运行得越快越好，真的要下一番功夫才能把这种欲望克制住。设计编程语言的时候，我们应该有意识地问自己，什么时候可以放弃一些性能，换来一点点便利性的提高。

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极其的时间才是真正的无效率。随着计算机速度越来越快，这会变得越来越明显。

未来相当漫长。如果我们想要减少语言内核中基本公理的数目，不妨把眼光放得远一点，想一想如果时间变量t趋向无限会怎么样。一百年是一个很好的参考指标，如果你觉得某个想法在一百年后仍然可能是难以令人接受，那么也许一千年后它也依然难以令人接受。

以函数库为例，它们具有可重用性，是因为它们属于语言的一部分，而不是因为它们采取面向对象或者其他编程方法。

除了某些特定的应用软件，一百年后，并行计算不会很流行。如果应用软件真的大量使用并行计算，这就属于过早优化了。

在任何学术领域，都有一些题目是可以做的，另一些题目是不可以做的。不幸的是，这两类题目的区别通常取决于它们写成论文后看上去是不是很高深，而不是取决于它们对软件业的发展是否重要。最极端的例子可能就是文学，文学研究者的任何成果几乎对文学创作者都毫无影响。

你也许觉得，对于人类这种懒惰的生物，喜欢用最省力的方式写程序是再自然不过的事情。但是事实上，我们的思想可能往往会受限于某种现存的语言，只采用在这种语言看来更简单的形式，它对我们思想的束缚作用会大得令人震惊。新语言必须靠你自己去发现，不能依靠那些让你自然而然就沉下去的思维定势。

学习开车的时候，一个需要记住的原则就是要把车开直，不是通过将车身对齐画在地上的分割线，而是通过瞄准远处的某个点。即使你的目标只在几米开外，这样做也是正确的。

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）

软件业是竞争非常激烈的行业，而且容易出现垄断。在不考虑其他情况的条件下，某家公司的软件更快更好用，就会把竞争者赶出这个市场。一旦你开始创业，你就会更深切地感受到这一点。一般情况是，创业公司要么赢得一切，要么彻底失败。你要么成为富翁，要么一无所获。创业的时候，如果你选择了错误的技术，竞争对手就会一举打败你。

既然你可以使用任何语言，你就不得不思考到底使用哪一种语言。如果你的公司对这种选择的自由视而不见，而竞争对手看到了，那么你就有被击败的危险。

如果从图灵等价的角度来看，所有语言都是一样强大的，但是这对程序员没有意义（没人想为图灵机编程）。程序员关心的那种强大也许很难正式定义，但是有一个办法可以解释，那就是有一些功能在一种语言中是内置的，但是在另一种语言中需要修改解释器才能做到，那么前者就比后者更强大。

除了某些特殊情况，你就是应该使用目前最强大的语言。不过在现实中这个结论很少能落实。到了一定年龄之后，程序员极少主动更换自己的编程语言。不管习惯使用的是哪一种语言，他们往往认为这种语言已经足够好了。

通过归纳法我们就会知道，唯一洞悉所有语言优劣的人必然是懂得最强大的那种语言的人。由于Blub困境的存在，你无法信任其他人的意见：他们都满足于自己碰巧用熟了的那种语言，他们的编程思想都被那种语言主宰了。

把宏说成一种独立的功能有误导之嫌。在实际运用中，如果没有其他Lisp功能（比如闭包和函数的rest参数）的配合，Lisp的宏业不会有太大作用。

编程语言的特点之一就是它会使得大多数使用它的人满足于现状，不想改用其他语言。人类天性变化的速度大大慢于计算机硬件变化的速度，所以编程语言的发展通常比CPU的发展落后一二十年。

技术的变化速度通常是很快的。但是，编程语言不一样，与其说它是技术，还不如说是程序员的思考模式。编程语言是技术和 宗教的混合物。所以，一种很普通的编程语言就是很普通的程序员使用的语言，它的变化就像冰山那样缓慢。

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”）

认为所有语言都一样的看法的缺点是自欺欺人，但是优点是可以使许多事情变得很简单。我想这就是为什么它被广泛接受的主要原因。它是一个令人舒服的想法。

大家都觉得Java一定有过人之处，因为它是一种很酷的新兴编程语言。但是真的如此吗？如果你站在远处观察编程语言的世界，似乎Java就是最新的东西。（如果你站得足够远，那么你看到的所有东西就是Sun公司出钱制作的大型霓虹广告牌。）但是，如果你靠近观察这个世界，就会发现不同的人对“酷”的理解是不一样的。

数学是不会过时的。你不应该把Lisp语言与50年代的硬件联系在一起，而是应该把它与快速排序算法进行类比。这种算法是1960年提出的，至今仍然是最快的通用排序方法。

有了服务器端软件能够解释为什么像Perl和Python这样的新语言会流行起来，它们之所以流行不是因为人们使用它们开发Windows应用程序，而是因为人们在服务器上使用它们。随着软件从桌面端向服务端转移，逼迫你使用某一种语言的限制将越来越少。

事实上，选择更强大的编程语言会减少所需要的开发人员数量。因为：(a)如果你使用的语言很强大，可能会减少一些编程的工作量，也就不需要那么多黑客了；(b)使用高级语言的黑客可能比别的程序员更聪明。

当我们只比较代码数量的时候，言下之意就是假设使用功能较弱的语言也能开发出同样的软件。但是事实上，程序员使用某种语言能做到的事情是有极限的。如果你想用一种低层次的语言解决一个很难的问题，那么你将会面临各种情况极其复杂乃至想不清楚的窘境。

不同语言的编程能力不一样。大多数经理故意忽视第一点。

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）

编程语言本来就是为了满足黑客的需要而产生的，当且仅当黑客喜欢一种语言时，这种语言才能成为合格的编程语言，而不是被当作“指称语义”或者编译器设计。

即使不考虑语言本身的优秀是否能带动流行，我想单单流行本身就肯定会使得这种语言变得更好，只有流行才会让它保持优秀。编程语言的最高境界一直在发展之中。虽然语言的核心功能就像大海的深处，很少有变化，但是函数库和开发环境之类的东西就像大海的表面，一直在汹涌澎湃。

实体书并没有过时，它们读起来很方便，而且出版社对书籍内容的审核是一种很有用的质量保证机制。书店则是程序员发现和学习新语言的最重要的场所之一。

有一个笑话说，黑客动手写程序之前，至少会在心里盘算一下哪种语言的打字工作量最小，然后就选择使用该语言。这个笑话其实与真实情况相差无几。

优秀程序员经常想做一些既危险又令人恼火的事情。所谓“令人恼火”，我指的是他们会突破设计者提供给用户的外部语义层，试着控制某些高级抽象的语言内部接口。

一种真正优秀的编程语言应该既整洁又混乱。“整洁”的意思是设计得很清楚，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”的意思是它允许黑客以自己的方式使用。

编程时提高代码运行速度的关键是使用好的性能分析器，而不是使用其他方法，比如精心选择一种静态类型的编程语言。

帕金森定律被证明与摩尔定律一样颠扑不破。软件不断膨胀，消耗光所有可以得到的资源。这一切将随着互联网软件的出现发生改变，因为硬件和软件现在捆绑在一起供应。

发明新事物的人必须有耐心，要长年累月不断地做市场推广，直到人们开始接受这种发明。

最终来看，自然成长式会比大爆炸式产生更好的技术，能为创始人带来更多的财富。如果你研究一下目前的主流技术，就会发现大部分都是源于自然成长式。

为了写出优秀软件，你必须同时具备两种互相冲突的信念。一方面，你要想初生牛犊一样，对自己的能力信心万丈；另一方面，你又要像历经沧桑的老人一样，对自己的能力抱着怀疑态度。

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。其他人看到他们的成果觉得棒极了，而创造者本人看到的都是自己作品的缺陷。这种视角的差异并非偶然，因为只有对现状不满，才会造就杰出的成果。

15 设计与研究（研究必须是“新”的，而设计必须是“好”的）

设计与研究的区别看来就在于，前者追求“好”，后者追求“新”。优秀的设计不一定很“新”，但必须是“好”的；优秀的研究不一定很“好”，但必须是“新”的。我认为这两条道路最后会发生交叉：只有应用“新”的创意和理论，才会诞生超越前人的最佳设计；只有解决那些值得解决的难题，才会诞生最佳研究。所以，最终来说，设计和研究都通向同一个地方，只是前进的路线不同罢了。

做一个好的设计师就像做一个好医生一样。你不能头痛医头，脚痛医脚。病人告诉你症状，你必须找出他生病的真正原因，任何针对病因进行治疗。大多数优秀设计都是这样产生的，它们关注用户，并且以用户为中心。

要记住一点，怎么理解编程语言？你不要把它看成那些已完成的程序的表达方式，而应该把它理解成促进程序从无到有的一种媒介。

士气也可以解释为什么很难为低端用户设计出优秀产品。因为优秀设计的前提是你自己必须喜欢这种产品，否则你不可能对设计有兴趣，更不要说士气高昂了。