我不是编译器专家

工作多年以来，我深刻体会到一个现象，那就是做过“编译器”工作的人，哪怕只做了点皮毛，都容易产生高人一等的心理，以至于在与人合作中出现各种问题。由于他们往往也存在偏执心理和理想主义，所以在恶化人际关系的同时，也可能设计出非常不合理的软件构架，浪费大量的人力物力。

我曾经提到的 DSL 例子，就是这样的两个人。他们都自称做过编译器，成天在我面前高谈阔论，甚至在最基础的概念上班门弄斧，显示出一副“教育”其他人的姿态。其实他们只有一个人做过 parser，还不算是真正的编译器工作，却总显示出高深莫测的模样。哲人一样捋捋胡子，摇摇脑袋，慢条斯理，嗯…… 另外一个完全就是外行，只是知道一些术语，成天挂在嘴边。每次他一开口，我都发现这个人并不知道他自己在说什么，却仍然洋洋得意的样子。

我是被他们作为专家请来这个公司的，来了之后却发现他们最喜欢的事情，是在我面前显示他们才是“专家”。他们也问过我问题，可是每一次我都发现他们并不想知道答案，因为我说话的时候他们并没有在听。不管说什么问什么，他们似乎只想别人觉得他们是最聪明的人。

虽然对其他人趾高气昂，全懂了的样子，对于 Brendan Eich（JavaScript 语言的创造者）这样有权势的人物，却是各种跪舔，显示出各种“贱”来。我虽然尊重 Brendan Eich 个人和他的语言，然而很明显他是半路出家，对语言设计并没有很深的造诣。对语言稍微有点研究的人，都不会对这种人物显示出谄媚的态度。

“Yin，你知道 X 吗？” 当然他期望的是你说不知道，这样他就能像大师一样，把这个刚学到的术语给你讲半天。每当这个时候，我就想起一个前同事喜欢说的一句话：“你问我，是因为你不知道，还是因为你知道？” 其实他问的这个概念 X，常常是我很多年前热心过，试验过，到最后发现严重问题，抛弃了的概念。

更糟的事情是，这其中一人还是 Haskell 语言的忠实粉丝，他总是有这样的雄心壮志，要用“纯函数式编程”改写全公司的代码……

遇到这样的人是非常闹心的，到了什么程度？他们经常雄心勃勃用一种新的语言（Scala，Go 之类）试图改写全公司的代码，一个月之后开始唾骂这语言，两个月之后他们的项目不了了之，代码也不知道哪里去了。然后换一种语言，如此反复……

后来实在没做出什么有用的东西，这两个人又突发奇想，开始做 DSL，闹得团队不得安宁，有点资历的工程师（包括我和一位早期 Netscape 的资深工程师）都极力反对，向大家指出更容易，更省力的解决方案。然而由于管理层根本不懂，所以任凭这两个人拍胸脯，没有困难制造困难也要上。因为烦于他们在我面前高谈阔论，而且对这个 DSL 的事情实在看不下去了，我干脆换了一个部门，不再做跟语言和编译器相关的事情。

现在这个 DSL 做了好几年了，仍然很垃圾，然而公司人傻钱多，居然请到了 Java 界的资深人物来给这 DSL 写 specification。这两人也分别升职为 Principal Engineer 和 Distinguished Engineer。当看到“Distinguished Engineer”这个 title，我觉得太好笑了。当然，我相信有资历的 PL 人都会明白这 DSL 的问题，我想象着这位 Java 人跟这两人将会发生的冲突。如果他对此没意见的话，那他的水平还真是值得怀疑了。

在 Coverity 和其它公司遇到的编译器人，基本是差不多的问题。他们下意识里把自己看成是最高档次的程序员，所以对其他人显示高高在上的气势。

Coverity 有一个 ABC 工程师，因为自己写过完整一点的静态分析，比较会折腾 C++，总是趾高气昂的对待其他人，甚至直接对别人说：“你写的这是什么代码啊？我绝对不会写出这么烂的代码！” 还有一个从斯坦福编译器教授 Alex Aiken 那里毕业的 PhD，在 Coverity 做构架师，平时一行代码不写，也不看其他人写的，说不出见解深刻点的话，因为与实际工程脱节，尽在瞎指挥。地位最高的 Distinguished Engineer，成天优哉游哉，看一些关于 parser 的话题，似乎 parser 是他终身的研究方向，也不做什么实事。

我所在的每一家公司，只要工作跟编译器沾边，总是不免遇到这样的人。其它的我就不细讲了。

有些美国公司在招人的时候表示，对简历里提到“做过编译器”的求职者有戒备心理，甚至直接说“我们不招编译器专业的人”。以至于我也曾经被过滤掉，因为我做过编译器相关工作。编译器专业的人本来可以做普通的程序员工作，为什么有公司如此明确不要他们呢？我现在明白为什么了，因为自认为是“编译器专业”的人，有大概率是性格很差的团队合作者，喜欢显示出高高在上，拯救世界的姿态，无法平等而尊重的对待其他人。

有些人也把我叫做“编译器专家”，喜欢在我面前提“编译器”这个词。我一直听着别扭，却没有正式拒绝这个称呼。每每遇到“真正”的编译器专家，我总觉得自己不是那个圈子的。不是我不能做编译器的工作，而是编译器领域人士的认识水平，理念和态度和我格格不入。

所以我应该明确表个态：我不是编译器专家，而且我看不起编译器这个领域。我一般不会居高临下看低其它人，然而对于认识肤浅却又自视很高的人，我确实会表示出藐视的态度。现在我的态度是针对编译器这整个领域。真的，我看这些人不顺眼很多年了。

就最后研究的领域，我是一个编程语言（PL）研究者，从更广的角度来看，我是一个计算机科学家。有人听了“科学家”一词总是误以为我在抬高自己，而在我心目中“科学家”仅仅是一个职业，就像“厨师”一样，并不说明一个人的水平和地位。PL 研究者被叫做“计算机科学家”是很恰当的，因为 PL 领域研究的其实不只是语言，而是计算的本质。通常

人公认的计算机科学鼻祖 Alan Turing 也可以算是一个 PL 研究者，虽然他认识水平比较一般。

IT 业人士经常混淆编程语言（PL）和编译器两个领域，而其实 PL 和编译器是很不一样的。真懂 PL 的人去做编译器也会比较顺手，而编译器专业的却不一定懂 PL。为什么呢？因为 PL 研究涵盖了计算最本质的原理，它不但能解释语言的语义，而且能解释处理器的构架和工作原理。当然它也能解释编译器是怎么回事，因为编译器只不过是把一种语言的语义，利用另外一种语言表达出来，也就是翻译一下。PL 研究所用的编程范式和技巧，很多可以用到编译器的构造中去，但却比编译器的范畴广阔很多。

深入研究过 PL 的人，能从本质上看明白编译器里在做什么。所以编译器算是 PL 思想的一种应用，然而 PL 的应用却远远不止做编译器。每次有人说我是做编译器的，我都觉得是一种贬低。我只不过拿精髓的理念稍作转换和适应，做了点编译器的事情，就被人叫做“编译器专家”，而我根本不是局限在这个方向。

专门做编译器的人，一般是专注于“实现”别人已经设计好的语言，比如 C，C++。他们必须按照语言设计者写好的语言规范（specification）来写编译器，所以在语言方面并没有发挥的空间，没有机会去理解语言设计的微妙之处。

许多做编译器的人并不是从零开始写的，而是拿现成的编译器来修改，所以他们往往被已经存在的，具体的构架限制了想象力。极少有编译器人完整实现过一个语言，都是在已有的基础上小改一下，优化一些局部的操作。这大大限制了他们可以获得的全局洞察力。

很多编译器工程师并没有接受过系统的 PL 理论教育，有些甚至是半路出家，在学校里根本没碰过编译器，也没研究过 PL。比如我的第一个公司 Coverity，招进去的很多人从来没碰过编译器，也不懂 PL。我进去不久，Coverity 的 VP 满口牛气向新人宣布：“我们会教会你们一切！”然而很可惜，PL 的精华根本不是一个公司在短期能够传授的。Coverity 没有这个能力，Google，Facebook，Intel，微软…… 都没有这个能力。

很多半路出家的编译器工作者以为在公司跟着做项目，折腾下 LLVM 之类，就会明白所有的原理。然而事实是很多人这样做了十几年，仍然不明白最基础的原理，因为他们被具体的实现限制了想象力。PL 理论联系着计算的本质，不明白这些原理就只能看到肤浅的表面，死记硬背，遇到新的现象就没法理解了。跟 LLVM 专家聊天，我很多时候发现他们的知识是死的，僵化在 LLVM 具体的实现里了。

由于缺乏对 PL 理论的深入研究，编译器人往往用井底之蛙的眼光来看待语言，总以为他们实现过的语言（比如 C++）就是一切。一个语言为什么那样设计？不知道。它还可以如何改进？不知道。“它就是那个样子！”这是我常听编译器人说的话。当然很多编译器人连 C++ 都没法完整实现，只是在已有基础上做了很小的改动。

许多编译器人把 C++ 的创造者 Bjarne Stroustrup 奉为神圣，却不知道 Stroustrup 在 PL 领域并不是闪耀的明星。Stroustrup 曾经在 2011 年 11 月 11 日来到 IU 进行关于 C++11 的演讲，IU 的资深 PL 教授们都有到场。Stroustrup 谦卑的说：“我需要向你们学习很多东西来改进 C++。”看似“谦虚”，其实他说的是实话，因为 IU 的教授们在语言设计上确实比他强很多。

Stroustrup 的整场演讲，我没有看到任何新颖的突破，全都是几十年早已出现，我天天都在用的东西。然而这些 C++ 的改进被编译器人看作是重大的历史性的突破，因为他们很多人根本没用过其它语言，甚至不知道它们的存在。

后来我的一个能力比较弱的 PL 同学进入了 C++ 委员会，为改进 C++ 做一些事情。从她的描述和表现，我感觉 C++ 委员会气氛十分的官僚，古板和愚钝。她进了 C++ 委员会之后，感觉整个人都傻了一样，很肤浅的小事也说得眉飞色舞，好像什么重大的突破一样。真懂 PL 的一些同学，很少有混进 C++ 委员会的，因为那意味着要利用另外的关系网，让一些自己根本看不起的人骑在自己头上，必须先帮他们做一些瞎扯淡的事情。

编译器人所膜拜的大师，在真正的 PL 研究者眼里其实不算什么。编译器人与 PL 研究者在见识上的差距是非常明显的。PL 人因为看透了很多东西，比较谦虚，往往不想揭穿编译器人的差距。但编译器人却因为在“工业界”有地位，趾高气昂以为自己懂了一切一样，结果遇到深刻点的 PL 问题就各种稀里糊涂。

实际上做编译器是很无聊的工作，大部分时候只是把别人设计的语言，翻译成另外的人设计的硬件指令。所以编译器领域处于编程语言（PL）和计算机体系构架（computer architecture）两个领域的夹缝中，上面的语言不能改，下面的指令也不能改，并没有很大的创造空间。

编译器领域几十年来翻来覆去都是那几个编程模式和技巧，玩来玩去也真够无聊的。起初觉得新鲜，熟悉了之后也就那个样了。很多程序员都懂得避免“低水平重复”，可是由于没有系统的学习过编译器，他们往往误以为做编译器是更高级，更有趣的工作，而其实编译器领域是更加容易出现低水平重复的地方，因为它的创造空间非常有限。

同样的编译优化技巧，在 A 公司拿来做 A 语言的编译器，到了 B 公司拿来做 B 语言的编译器…… 大同小异，如此反复。运气好点，你可能遇到 C，C++，Java。运气不好，你可能遇到 JavaScript，PHP，Ruby，Go 之类的怪胎，甚至某种垃圾 DSL。但公司有要求，无论语言设计如何蹩脚，硬件指令设计如何繁琐，你编译出来的指令必须能正确运行所有这语言写出来的代码。你说这活是不是很苦逼？

我在 Cornell 的时候，有一个很有权势的编译器教授，从未发表有理论价值的 paper，却老在 Java 上面做文章。他和他的博士生们总是把一些其它语言几十年前已经有的“新特性”搬到 Java 上面，老酒换新瓶，发 paper 拉 funding。由于拉了很多钱，所以在系里很受宠，他的学生们在其它人面前都趾高气昂的样子。

后来这教授的一个学生去了 Facebook，帮他们做 HipHop，一个从 PHP 到 C++ 的“编译器”。其实这种“源到源”编译器做起来不算难，但给 PHP 这样劣质的语言做编译器，实在是狗血的工作，繁琐而头痛。没有任何理论价值

不说，在工业界有什么价值也难说。我的一个前同事曾经对 Facebook 的这个项目发表了一个尖锐而幽默的评价：“Facebook 现在不但给母猪涂上了口红，而且真的开始 f.. 它了！”

后继的还有 PHP VM 一类的东西，越来越离谱。后来这位同学可能也受不了，换组去做其它跟语言无关的事情了。在 PL 研究者看来，VM 也并没有什么稀奇。PL 领域有各种各样的“抽象机”（abstract machine），比如 CEK machine，它们揭示了计算的方方面面。我自己都设计实现过好几个“可逆抽象机”，它们可以进行所谓“可逆计算”。所以一个 PL 研究者很容易就能设计出一个 VM 来，它们只不过是一种经过部分优化的解释器。

每每看到编译器人说到“VM”这个词的时候那种荣耀而敬畏的神情，好像只有他们明白 VM 是什么，我就觉得好笑，外加一种说不出的滋味。编译器人虽然知道一个具体的 VM 怎么实现，知道一些死板的细节和术语，却不知道 VM 的本质是什么，不知道一个全新的，具有新特性的 VM 要怎么设计出来。

在《Chez Scheme 的传说》一文中，我提到在 Cornell 的时候选过一门编译器课程，后来在半学期的时候 drop 掉了。现在回想起这段历史，发现它对“教育理念”这件事挺有启发意义。教育是什么，是为了什么？Cornell 的这门课给了我一个很好的反面教材。

这个编译器课程那一年的教授是 Tim Teitelbaum，他也是 GrammaTech 公司的创始人。GrammaTech 是与 Coverity 类似的静态分析工具，不过 GrammaTech 还能分析二进制代码。Tim Teitelbaum 是 Donald Knuth 的崇拜者，他经常提到 Knuth 提出的一些“伟大概念”，比如 attribute grammar。总是把 Knuth 那些东西说成是最伟大的发明。

这门课不知道最初是谁设计的。Andrew Myers 和 Tim Teitelbaum 以前交替着讲这个课。

那么我为什么会 drop 这门课，而且是在学校允许 drop 课程的 deadline 之后呢？因为它的教育理念非常的落后和不合理，可以说就是坑人的。

从课程的大纲你可以看出来，它是很传统的编译器课程，一开头花很多时间精力去折腾 parser。源语言是一种类似 Java 的语言，parser 是使用类似 lex，yacc 的工具生成的。这种盲目重视 parser 的误区，我已经在另外一篇文章批评过，但还这不是我鄙视的重点。

这门课最让人受不了的事情，发生在我成功完成 parser，开始编译代码的第一个 pass 之后。当得到那次作业分数的时候，我惊呆了。我从来没有得过这么差的分数！仔细看原因，说我的代码没通过好些“测试”。我到那个时候才明白，原来提交后的代码，会被助教拿来跑一些我毫不知情的测试（test），然后他简单的根据这些测试的结果给出分数。

作业本身的要求是用大段大段的英语写下来的。你需要按照这些英语描述从零实现编译器。真的是从零开始，没有任何的框架或者示例代码，完全从白纸开始。经过许多努力，你写出了编译器，还自己写了一些小测试，你觉得完全满足了作业的要求。可是提交之后，你的编译器代码却要被一整套你手里没有的“测试”进行检验。所以最后你惊讶的发现，自己以为做对了，而助教那里的测试有那么多没通过！

最让人无语的事情是，学生手里是没有这套测试的，而且他们不给你。也就是说，你提交作业的时候，无法用最后给你评分用的那些测试来跑你的编译器，所以你无法知道提交之后会有多少测试失败。

当我向助教和教授抗议，说这样不合理，要求得到那些测试的时候，我受到粗暴的拒绝和鄙视。那种语气，好像是在说我是一个不合格的学生，提一些无理要求。用来打分的测试怎么可能给你，你是太笨了吧？

很多其它 Cornell 学生被这样对待，可能都以为没什么，按照他们的要求做就行了，然而这是完全不合理的。按照合理的教学理念，学生应该有权得到自己学习状态的反馈。如果学生做这种编程作业，就应该能从实际的测试中得到反馈，知道自己的编译器是否符合要求。要知道，大段大段的英语描述，是很容易漏看或者误解的。只有大量的测试才能正确的抓住“要求”本身。所以不给测试，就相当于不给你准确的要求，到后来却要拿这套测试来给你打分。

课程本来应该把测试连同英语描述一起给学生，他们实现之后，自己跑通所有测试，再提交代码。这样学生就能准确的把握作业的“要求”，而不是看着那些混淆不堪的英语段落自己在那里猜。

因为这个原因，而且由于教授和助教的傲慢态度。我最终决定在课程都快进行到一半的时候 drop 这门课程。当然，要进行这个操作是需要系主任签字特许的，为此我还在系主任那里留下一笔“污点”。

在我看来，Cornell 教授们的这种做法，根本就不是合格的教育者，可以说就是在坑人，整人，害人。在他们的理念里，教育是单方面的，学生必须通过作业和考试，而教授却不需要为教学方法负责，可以随便怎么教，作业和考试想怎么整都行。

很多 Cornell 教授有类似的现象，教学不用心，光是各种拉 funding，耀武扬威，完全不顾学生死活。也许这就是为什么 Cornell 总是有学生自杀。我走了之后有一年，在一个星期之内有三个学生从学校里瀑布旁边的吊桥跳下去自杀，新闻轰动了全美国。

后来在网上看到有人骂 Cornell，说：“Cornell 想教你游泳，于是他把你推进池塘里，等你扑腾上岸。等你快上来的时候，他又朝你扔一块大石头，然后继续等你游上来。等你又快上岸了，他又拿起一个榔头往你头上猛砸。这样你就可以死了，可是 Cornell 仍然在那里等着你游上岸来……”

这段话恰到好处的描述了我的在 Cornell 的经历。

转学到 IU 之后，我参加了 Kent Dybvig 的编译器课程，发现我所设想的编译器课程原来早已被他实现了，而且实现的如此友好。编译器的每一个 pass，都会把所有的“官方测试”发给学生。学生按照要求实现每个编译器 pass，在自己电脑上跑通所有测试，充分检查，然后才提交作业。而且作业的网站会自动测试你提交的代码，在提交的当时就给你反馈：“你有 N 个测试没通过，请修改后重新提交。”

这才是正确的教育方法，因为它给予学生合理的反馈，让他们清晰的知道自己的表现是否符合预期，主动进步，而不是拿一些学生事先不知道的标准在那里瞎坑人，光是给人打分。

Cornell 没有明白教育的目的是培养人，而不只是给人发文凭。Dybvig 教授不但技术和学术水平远高于传统的编译器人，而且他的课程也设计得如此科学和友好。这才是真正的教育者。

虽然苦逼，编译器人往往自高自大，高估自己在整个 IT 领域里的地位，看低其它程序员。编译器人很多认为自己懂了编程语言的一切，而其实他们只是一知半解。

编译器领域最重要的教材，龙书和虎书，在我看来也有很多一知半解，作者自己都稀里糊涂的内容。而且花了大量篇幅讲 parser 这种看似高深，实则肤浅的话题，浪费读者太多时间，误导他们认为 parser 是至关重要的技术。以至于很多人上完编译器课程，只学会了写 parser，对真正关键的部分没能理解。龙书很难啃，为什么呢，因为作者自己都不怎么懂。虎书号称改进了龙书，结果还是很难啃，感觉只是换了一个封面而已。

我曾经跟虎书作者 Andrew Appel 的一个门徒合作过，当时这人在 IU 做助理教授。借着一次我跟她做 independent study 的机会，逼我写毫无意义的论文，而且对人非常的 push 和虚伪。作为普林斯顿大学毕业的 PhD，学识水平跟 IU 的其他教授格格不入，却在待人接物方面显示出各种“贱”，对编译器领域的“牛人”各种跪舔，随时都在显示自己以前在某某人身边工作过。那是我在 IU 度过的最难受的一个学期，这使我对“编译器人”的偏见又加深一层。

编译器领域的顶级人物如此，其它声称做过编译器的人也可想而知了。大部分自称做过编译器的人，恐怕连最基本的的编译器都没法从头写出来。利用 LLVM 已有的框架做点小打小闹的优化，就号称自己做过编译器了。许多编译器人士死啃书本，肤浅的记忆各种术语（比如 SSA），死记硬背具体实现细节（比如 LLVM 的 IR），看不透，无法灵活变通。

所以我常说，编译器是计算机界死知识最多，教条主义最严重的领域。经常是某人想出一个做法，起个名字，其他人就照做，死记硬背，而且把这名字叫得特别响亮。你要是一时想不起这名字是什么意思，立马被认为是法国人不知道拿破仑，中国人不知道毛泽东。你不是做编译器的！

现在因为 AI 的泡沫，很多人转向所谓“AI 框架”，“AI 编译器”。这类职位如此之多，以至于很多人根本没碰过编译器，也摇身一变成为了“深度学习编译器工程师”。

半路出家的“AI 框架工程师”和“AI 编译器工程师”们，在别人写出来的框架上小打小闹优化一下，就以为自己做的是世界上最前沿的工作，却不知道深入研究过 PL 的人其实很容易就看破了那些东西。很多 AI 框架工程师嘴里各种奇怪的术语，却看不透所谓“AI 框架”只不过是“可求导编程语言”，完全不能从高级语言和逻辑的角度去看问题。

AI 框架和编译器里面的原理和本质很容易被 PL 理论解释，PL 研究者能够为这些项目指出正确的方向，避免不必要的弯路，然而这些自诩为“编译器人”的 AI 框架工程师们完全意识不到这一点。自高自大，膜拜权威，完全没有去听 PL 研究者在说什么，甚至觉得能“教育”比自己看得透的人。

每一个大公司都要趁着 AI 这个热度做自己的“AI 框架”，“AI 编译器”，唯恐不做自己的框架，就会在业界丢面子，所以一窝蜂而上。一定要聘用名声很大的 AI 框架专家来公司站台，虽然也不知道他最后能做出什么来。所有 AI 框架和编译器都大同小异，属于无谓的重复劳动。有些人捣鼓一下这个框架，然后用同样的技巧去捣鼓另外一个，中间都是一些工程性的脏活。这种事情真是非常无聊。

AI 的热潮正在褪去，大部分 AI 公司会在一年之内失败。“AI 编译器”的工作也会濒临灭绝。所以任凭他们自己瞎蒙乱撞吧，反正坚持不了多久了。

这就是为什么虽然有多次编译器的工作机会，包括 Apple 的 LLVM 部门，我最后都没去。进入 Intel 的时候，本来编译器部门也欢迎我，可是再三考虑之后还是选择了其它方向。因为我很清楚的记得，每一次做编译器相关工作都是非常压抑的，需要面对一些沉闷古板而自以为是的人，而且内容真的是重复，无聊和枯燥。

我唯一敬佩的编译器作者是 Kent Dybvig，但我也不想跟他一起做编译器。最近很多芯片公司的“AI 编译器”部门找我，我全都拒绝了。我不喜欢身边围绕着这些人，做着这些事。我宁愿去卖烧饼也不想做编译器。

由于编译器人的性格特征，除非一个公司专门要做编译器，否则对于曾经做过编译器，想换个方向的求职者，在面试的时候最好深刻了解他们的性格，态度和做事方式，看他们是否能看淡这些，能否平等对待其他人，能否理性而实在的对待工程。否则自视很高的“编译器人”进了公司，很可能对团队成为一种灾难。

我写这篇文章是为了警醒广大 IT 公司，也是为了在精神上支持其它程序员。我希望他们不要被编译器的“难度”迷惑了，不要被编译器人吓唬和打压。你们做的并不是更低级，更无聊的工作。正好相反，真正可以发挥创造力的空间并不在底层的编译器一类的东西，而在更接近应用和现实的地方。

每当有人向我表示编译器高深莫测，向往却又高攀不上，我都会给他打一个比方：做编译器就像做菜刀。你可以做出非常好的菜刀，然而你终究只是一个铁匠。铁匠不知道如何用这菜刀做出五花八门，让人心旷神怡，米其林级别的菜肴，因为那是大厨的工作。要做菜还是要打铁，那是你自己的选择，并没有贵贱之分。