论“自然语言编程”的不智*

Edsger W. Dijkstra

1978 年

ChatGPT (译)
苑明理† (校)

【译者按】在 *2023* 年末的当代，我们讨论 *1978* 年一位学者论述的意义在哪里？一方面 *OpenAI* 带来的 *ChatGPT* 通过自然语言的提示语工程已经加速了很多行业的运转；另一方面，*Dijkstra* 在计算机科学领域的贡献如此卓著，才使得今天许多基础软件稳定运行。前者的可能性中一部分建立在后者的基础上。表面上，*Dijkstra* 的论点和现实是矛盾的，但背后的意义仍然值得深入思考。

从自动计算的早期开始，总有人认为编程必须如同使用任何形式符号体系那样的精确与谨慎是一种缺陷。他们批评机器对于其执行的指令过于严格的服从性，哪怕稍加思考就能发现这些指令明显错误。正如 A.E.Houseman 所说：“但一刻钟对于思考来说是很长的时间，而思考本身是个痛苦的过程。”他们热切地期待并寻找那些更智能的机器，它们能拒绝执行明显愚蠢的任务，如小错误引发的活动。

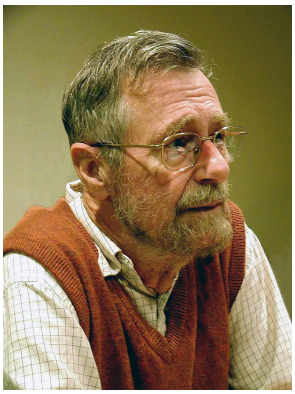

图 1: Edsger W. Dijkstra

机器代码，由于几乎不包含任何冗余，很快就被认为是人与机器之间不必要的风险界面。部分作为对

*原文位于 E. W. Dijkstra Archive 编号 667 手稿，原题目On the foolishness of ”natural language programming”；本译文的插图均由译者添加，图片来自维基媒体共享资源网站

这种看法的回应，所谓的“高级编程语言”应运而生，随着时间的推移，我们学会了在一定程度上增强了对愚蠢错误的防护。一个重要的进步是，现在许多愚蠢的错误将导致错误信息的产生，而不是错误的结果。（甚至这种改进并不是普遍受欢迎：有些人发现无法忽略的错误信息比错误的结果更令人烦恼，而在评价编程语言的相对优势时，一些人仍然倾向于将“编程的简易度”与犯下未被察觉错误的容易度相等同。）然而，与编程语言相对应的（抽象）机器仍然是一个忠实的奴隶，即那种完全能够执行无意义指令的非理性自动机。编程仍然是使用形式化的符号体系，因此，依旧需要之前所要求的那种精确和谨慎。

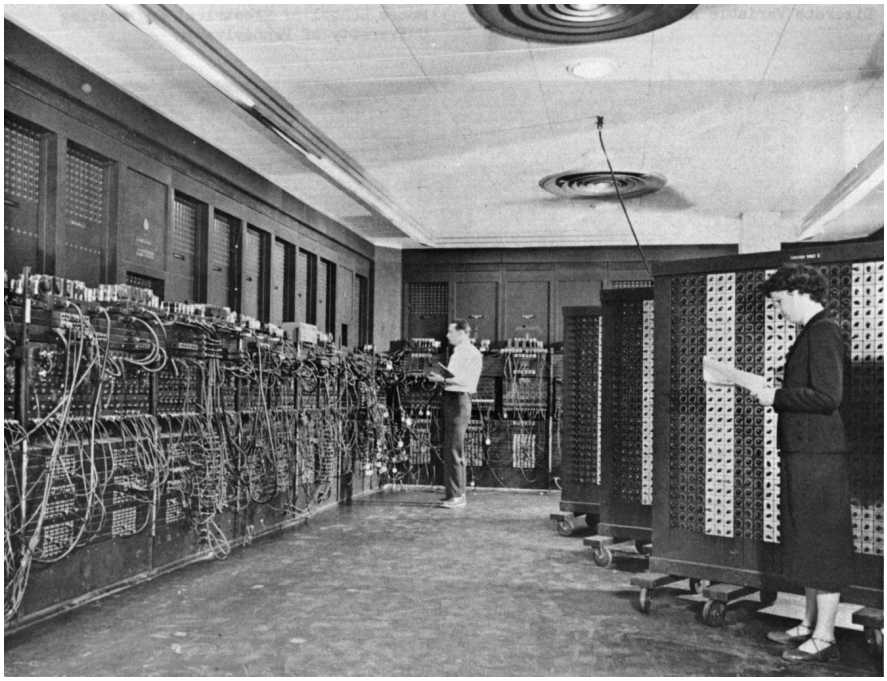

图 2：世界上第一台计算机 ENIAC

为了让机器使用更加简便，有人提出了一个想法：（尝试）设计可以接受我们用母语发出的指令的机器。诚然，这会让机器变得更复杂，但有人辩称，通过让机器承担更重的负担，我们的生活将变得更轻松。如果你认为使用形式符号体系是造成困难的根源，这看起来似乎合理。但这个论点真的成立吗？我对此表示怀疑。

我们现在明白，选择一个界面不仅仅是（固定数量的）工作分配问题，因为还必须考虑界面上进行协作和交流所涉及的工作。根据我们的实际经验—我不得不说，这是一种警醒的体验—我们知道，改变界面很容易在界面两侧都增加工作量（有时甚至是极大的增加）。因此，现在更倾向于使用所谓的“狭窄界面”。所以，变成用人类的母语进行机器与人的交流会大幅增加机器的负担，我们还是不得不对这样做会简化人类生活的假设提出质疑。[1]

[1] 【译者按】为什么在计算机发展的早期会是“狭窄界面”，而如今“宽泛界面”就成为可能呢？需要对 Dijkstra 的质疑本身提出质疑，只有分析清楚背后的原因，才能更清楚地理解未来的道路

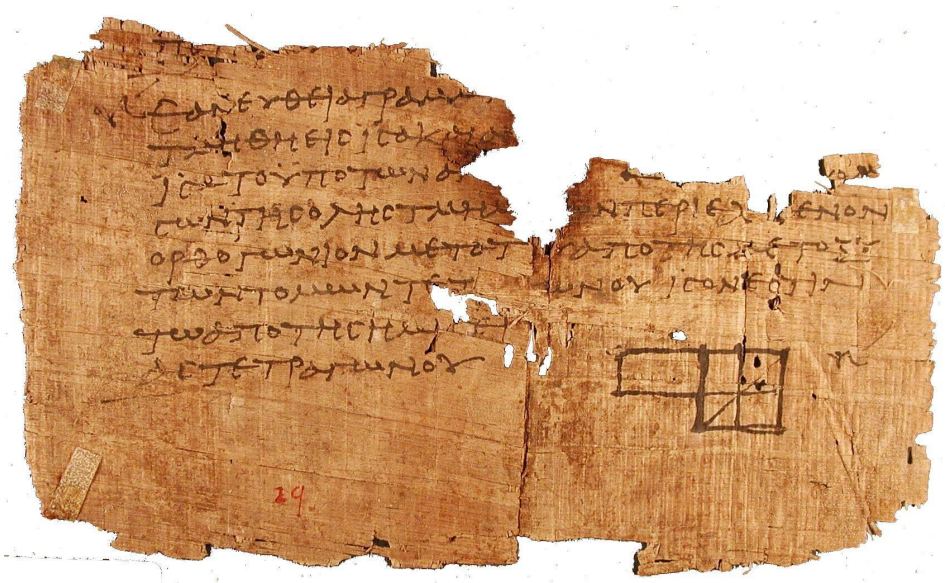

图 3：《几何原本》残片

简单回顾数学史就能证明这种挑战有多么有道理。希腊数学之所以停滞不前，是因为它一直停留在口头和图像化的层面上；穆斯林的“代数”在尝试使用符号之后变得胆怯，当它回归修辞风格时便走向了衰亡。现代文明世界能够出现—无论是好是坏—是因为西欧能够摆脱中世纪经院哲学的枷锁—那是对语言精确性的徒劳尝试！—这要归功于像 Vieta、Descartes、Leibniz 和（后来的）Boole 等人精心设计或至少是有意识设计的形式符号。

图 4：Vieta、Descartes、Leibniz 像

形式文本的优点在于，它们的操作要合法，只需遵守一些简单的规则。仔细想想，你会发现它们是一种极其有效的工具，能够排除各种荒谬，这些荒谬在我们使用母语时几乎是不可避免的。

我们不应将使用形式符号看作一种负担，反而应将使用它们的便利视为一种特权：得益于它们，学生们能够学会做过去只有天才才能完成的事情。（显然，这点并没有被 1977 年在一份技术报告前言中写到的作者理解，他甚至为了所谓的“清晰”，避免使用逻辑连接符的标准符号。这句话的出现表明，这种误解并不仅限于他一人。）总而言之，

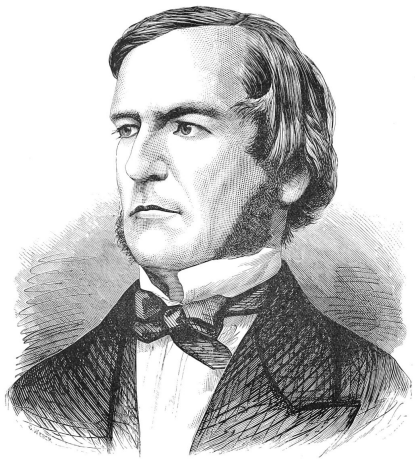

图 5: George Boole 像

我们使用母语的“自然”实质上就是我们轻易地使用它们来发表那些荒谬的言论的能力。[2]

尝试想象一下，如果从一开始我们的母语就成为我们信息处理设备的唯一输入和输出方式，可能会有一些启发。我经过深思熟虑的猜想是，历史在某种意义上可能会重演，计算机科学主要将是一种确实深奥的艺术，即如何从那里引导出一个定义足够明确的形式系统。我们将需要动用全世界的智力来使界面足够窄，从而使其可用，并且考虑到人类的历史，猜测要做到足够好可能还需要几千年的时间，这并不是过于悲观的想法。

我从一种强烈的直觉中获得了许多安慰：我怀疑，无论是用荷兰语、英语、美式英语、法语、德语还是斯瓦希里语来编程的机器，它们的制造难度和使用难度都同样巨大。

Plataanstraat 5
5671 AL NUENEN
荷兰 Edsger W.Dijkstra 教授博士
Burroughs 研究员

[2] 【原作者按】由于教育趋势偏离了智力训练，过去几十年来，在西方世界，人们对自己语言的掌握程度显著下降：许多按上一代标准来看应该更懂得的人，现在已经无法有效使用自己的母语，即使是在它相当适用的情况下也是如此。（你只需要看看科学文章、技术报告、政府出版物等中那些令人担忧的、在仔细阅读后显得毫无意义的冗长文字。）这一现象—被称为“新文盲”—应当让那些缺乏足够技术洞察力来预测其失败的自然语言编程的信徒们感到泄气。

A Edsger W. Dijkstra 小传

Edsger W. Dijkstra，1930 年 5 月 11 日出生于荷兰，是一位杰出的计算机科学家。他在 Amsterdam 大学学习物理和数学，后专注于计算机科学。Dijkstra 的职业生涯始于荷兰的数学中心，并在包括 Eindhoven 技术大学和美国 Texas 大学 Austin 分校等多所大学任教。他因其在编程方法论、操作系统、并发程序设计方面的研究而闻名。

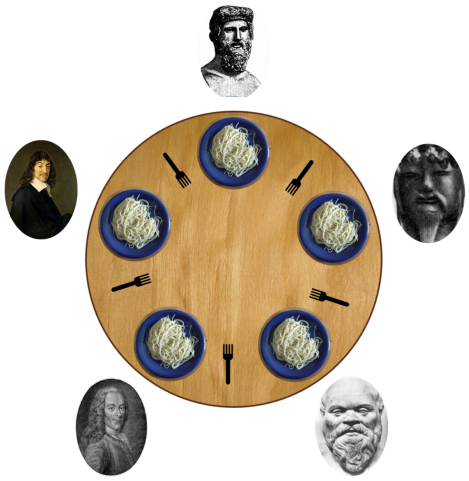

图 6: 由 Dijkstra 提出的哲学家晚餐问题

Dijkstra 的主要成就包括创造著名的短路径算法，该算法在网络路由和地图服务中被广泛应用。他是结构化编程的坚定支持者，反对使用 GOTO 语句，这一观念对现代编程实践和教育产生了重大影响。在操作系统领域，Dijkstra 引入信号量的概念，对进程同步和死锁预防做出了突出贡献。此外，他提出了 Communicating Sequential Processes（CSP）模型，极大地推动了并发编程理论和实践的发展。

Dijkstra 还开创了程序的形式方法领域，强调使用数学和逻辑方法表达和验证程序的正确性。这些方法对软件工程至关重要，尤其是在高可靠性和安全关键系统的设计中。

他的成就获得了广泛认可，包括 1972 年获得的图灵奖。Dijkstra 的工作不仅在他所处的时代具有划时代的意义，而且在当今计算机科学和软件工程领域仍具有深远的影响。2002 年 8 月 6 日，Dijkstra 在荷兰逝世，但他的理念和研究仍然影响着计算机科学的发展方向。