**胸有成竹（Calling the Shot）**

Frederick P. Brooks

**胸有成竹（Calling the Shot）**

实践是最好的老师。

- PUBILIUS

实践是最好的老师，但是，如果不能从中学习，再多的实践也没有用。

- 《可怜的理查年鉴》

*Practice is the best of all instructors.*  
—*PUBLILIUS*

*Experience is a dear teacher, but fools will learn at no other.*  
—*POOR RICHARD'S ALMANAC*

**编程出一个系统需要花费多长的时间？需要多少的工作量？如何进行估计？**

我之前提出的比例似乎适用于计划时间、编码、组件测试和系统测试。首先，需要指出的是，我们不能仅仅通过估计编码部分然后应用比率来估计整个任务。编码只占了问题的六分之一左右，在估计或比率上的错误可能导致荒谬的结果。

第二，必须指出，用于构建**孤立的小程序**的数据不适用于编程系统产品。

对规模平均为3200个单词的程序，比如说，如Sackman、Erikson和Grant的报告中所述，一个程序员所需要的编码和调试时间约为178个小时，这个数字可以外推得到每年35,800语句的生产率。

**2024年4月23日16:16:08：**

ChatGPT：

要计算每年的生产率，即每年可以编写多少行代码，我们可以从提供的每个程序编写和调试时间出发。根据Sackman、Erikson和Grant的报告，编写和调试一个平均3200行代码的程序需要178个小时。我们可以用这个信息来计算年生产率。

首先，我们需要确定一个年工作小时数。通常情况下，一个全职工作的程序员每年的工作小时数可以这样计算：

* 每周工作小时数通常是40小时。
* 一年通常工作周数是52周。

因此，一年的工作小时数大约是：

**40 小时/周×52 周/年=2080 小时/年40 小时/周×52 周/年=2080 小时/年**

接下来，我们使用这个信息来计算年生产率：

* 编写和调试3200行代码需要178小时。
* 每小时可以编写的代码行数是：

**3200 行178 小时≈17.98 行/小时178 小时3200 行​≈17.98 行/小时**

然后，使用每小时的代码行数来计算全年可以编写的代码行数：

**17.98 行/小时×2080 小时/年≈37,398 行/年17.98 行/小时×2080 小时/年≈37,398 行/年**

这个计算得到的结果是37,398行代码每年，这个数字与提供的35,800行代码每年略有差异。这种差异可能是由于不同的工作时数设定、假期、病假等因素影响实际工作时间，或者原始报告中使用了不同的平均工作时间计算。

因此，35,800行代码可能是基于稍微不同的每周工作小时数或者实际工作周数的计算结果。如果我们调整这些数字，可能会得到接近35,800行的结果。例如，如果实际工作时间略少于2080小时/年，那么计算出的年生产率会接近35,800行。

而规模只有一半的程序花费时间大约仅为前者的四分之一，而推断出的生产率几乎是每年80,000代码行1。

**1.** Sackman, H., W. J. Erikson, and E. E. Grant, "Exploratory experimentation studies comparing online and offline programming performance," *CACM*, **11**, 1 (Jan., 1968), pp. 3–11.

计划、编制文档、测试、系统集成和培训的时间必须被考虑在内。

**2024年4月23日16:26:22：上面内容讲的意思是，我们有两个事实：**

**事实1：3200个单词的小程序需要178小时来完成，如果按3200行/178小时这个速度来算，一个程序员一年可以搞出35,800行代码。**

**事实2：如果我们有1600个单词的小小程序需要完成，我们可以推断出一个程序员一年可以搞出80,000行代码。**

**当程序很小（1600单词）的时候，我们可以以80,000行代码/每年的编程速度工作。**

**但是当程序增大一倍（3200单词）的时候，我们的编程速度骤降到35,800行代码/每年。下降了2.23倍。**

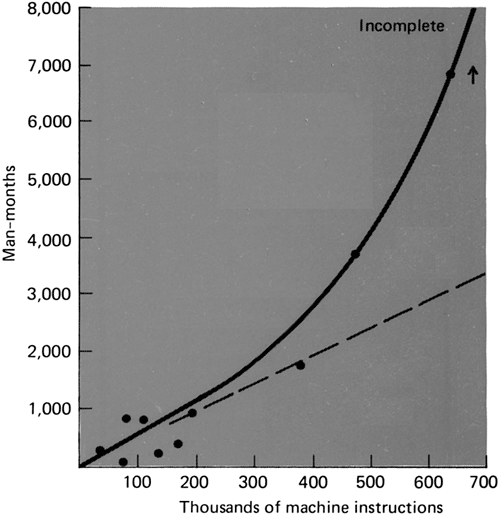
**然而系统级的超大规模编程项目至少有几万个单词。鬼知道他会下降到多少行代码每年的速度。（待会你就知道了，事实证明，大概是这个公式：**工作量 ＝ (常数)×(指令的数量)1.5**）**

**你可想而知，如果我们还用1600行代码的小程序的编程速度（80,000行代码/每年）去预测超大规模编程的项目进度，那我们只能死的很惨。**

**刻舟求剑。**

因此，上述小型项目数据的外推是没有意义的。就好像把100码短跑记录外推，得出人类可以在3分钟之内跑完1英里的结论一样。

在将上述观点抛开之前，尽管不是为了进行严格的比较，我们仍然可以留意到一些事情。即使在不考虑相互交流沟通，开发人员仅仅回顾自己以前工作的情况下，这些数字仍然显示出工作量是规模的幂函数。



横轴：机器代码（单位：以1000个单词）

**图8.1：编程工作量是程序规模的函数**

注释: incomplete－未终结的

图8.1讲述了这个悲惨的故事。它阐述了Nanus和Farr2在System Development Corporation公司所做研究，结果表明该指数为1.5，即，

工作量 ＝ (常数)×(指令的数量)1.5

effort = (constant) x (number of instructions)1.5.

Weinwurm[3]的SDC研究报告同样显示出指数接近于1.5。

**3.** Weinwurm, G. F., "Research in the management of computer programming," Report SP-2059, System Development Corp., Santa Monica, 1965.

现在已经有了一些关于编程人员生产率的研究，提出了很多估计的技术。 Morin对所发布的数据进行了一些调查研究[4]。这里仅仅给出了若干特别突出的条目。

**4.** Morin, L. H., "Estimation of resources for computer programming projects," M. S. thesis, Univ. of North Carolina, Chapel Hill, 1974.

**Portman's Data**

曼彻斯特Computer Equipment Organization（Northwest）的ICL软件部门的经理Charles Portman，提出了另一种有用的个人观点[5]。

**5.** Portman, C., private communication.

他发现他的编程队伍落后进度大约1/2，每项工作花费的时间大约是估计的两倍。

这些估计通常是非常仔细的，由很多富有经验的团队完成。

他们对PERT图上数百个子任务估算过（用人小时作单位）。

ChatGPT：

PERT（Program Evaluation Review Technique）图是一种用于项目管理的工具，主要用于规划和调度项目活动。PERT图是一种网络图，用于描述和可视化项目中各个任务的顺序、依赖关系和预期的持续时间。

当偏移出现时，他要求他们仔细地保存所使用时间的日志。日志显示事实上他的团队仅用了百分之五十的工作周，来进行实际的编程和调试，估算上的失误完全可以由该情况来解释。其余的时间包括机器的当机时间、高优先级的无关琐碎工作、会议、文字工作、公司业务、疾病、事假等等。简言之，项目估算对每个人年的技术工作时间数量做出了不现实的假设。我个人的经验也在相当程度上证实了他的结论[6]。

**6.** An unpublished 1964 study by E. F. Bardain shows programmers realizing 27 percent productive time. (Quoted by D. B. Mayer and A. W. Stalnaker, "Selection and evaluation of computer personnel," *Proc. 23rd ACM Conf.*, 1968, p. 661.)

**Aron's Data**

Joel Aron，IBM在马里兰州盖兹堡的系统技术主管，在他所工作过的9个大型项目（简要地说，大型意味着程序员的数目超过25人，将近30,000行的指令）[7]的基础上，对程序员的生产率进行了研究。

**7.** Aron, J., private communication.

他根据程序员（和系统部分）之间的交互划分这些系统，得到了如下的生产率：

**非常少的交互 10,000指令每人年**

**少量的交互 5,000**

**较多的交互 1,500**

该人年数据未包括支持和系统测试活动，仅仅是设计和编程。当这些数据采用除以2，以包括系统测试的活动时，它们与Harr的数据非常的接近。

**Harr's Data**

John Harr，Bell电话实验室电子交换系统领域的编程经理，在1969年春季联合计算机会议[8]的论文中，汇报了他和其他人的经验。

这些数据如图8.2、8.3和8.4所示。

**8.** Paper given at a panel session and not included in the *AFIPS Proceedings*.

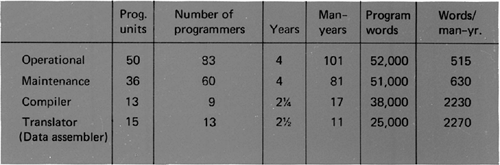


Figure 8.2. Summary of four No. 1 ESS program jobs

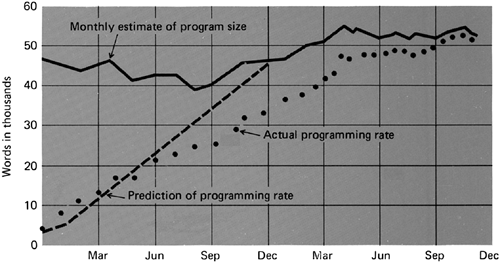


Figure 8.3. ESS predicted and actual programming rates

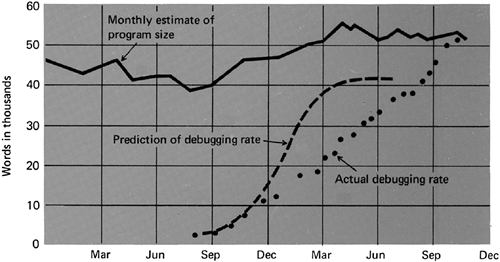


Figure 8.4. ESS predicted and actual debugging rates

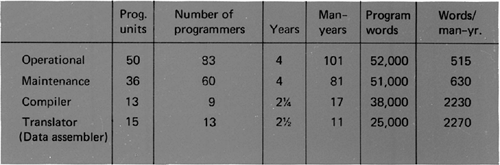


Figure 8.2. Summary of four No. 1 ESS program jobs

这些图中，图8.2是最数据详细和最有用的。头两个任务是基本的控制程序，后两个是基本的语言翻译。生产率以经调试的指令/人年来表达。它包括了编程、构件测试和系统测试。没有包括计划、硬件机器支持、文书工作等类似活动的工作量。

生产率同样地被划分为两个类别，控制程序的生产率大约是600指令每人年，语言翻译大约是2200指令每人年。注意所有的四个程序都具有类似的规模--差异在于工作组的大小、时间的长短和模块的个数。

那么，哪一个是原因，哪一个是结果呢？

是否因为控制程序更加复杂,所以需要更多的人员？

或者因为它们被分派了过多的人员，所以要求有更多的模块？

是因为复杂程度非常高，还是分配较多的人员，导致花费了更长的时间？

没有人可以确定。

**控制程序确实更加复杂**。除开这些不确定性，数据反映了实际的生产率--描述了在现在的编程技术下，大型系统开发的状况。因此，Harr数据的确是真正的贡献。

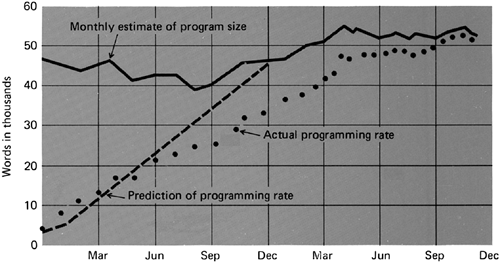


Figure 8.3. ESS predicted and actual programming rates

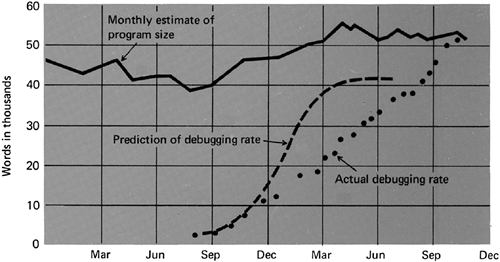


Figure 8.4. ESS predicted and actual debugging rates

图8.3和8.4显示了一些有趣的数据，将实际的编程速度、调试速度与预期做了对比。

**OS/360 Data**

IBM OS/360的经验，尽管没有Harr那么详细的数据，但还是证实了那些结论。就控制程序组的经验而言，生产率的范围大约是600～800（经过调试的指令）/人年。语言翻译小组所达到的生产率是2000～3000（经过调试的指令）/人年。这包括了小组的计划、代码构件测试、系统测试和一些支持性活动。就我的观点来说，它们同Harr的数据是可比的。

Aron、Harr和OS/360的数据都证实，生产率会根据任务本身复杂度和困难程度表现出显著差异。在复杂程度估计这片"沼泽"上的指导原则是：编译器的复杂度是批处理程序的三倍，操作系统复杂度是编译器的三倍[9]。

**9.** Wolverton, R. W.; "The cost of developing large-scale software," *IEEE Trans. on Computers*, **C-23**, 6 (June, 1974) pp. 615–636. This important recent paper contains data on many of the issues of this chapter, as well as confirming the productivity conclusions.

**Corbatò's Data**

Harr和OS/360的数据都是关于**汇编语言**编程的，关于使用高级语言进行系统编程的生产力，似乎很少有数据发表。

**2024年4月23日17:12:18：哦哥们，汇编语言，我的天哪。**

然而，麻省理工学院 MAC 项目的 Corbatò 报告说，在**MULTICS系统**上，每人年的平均生产率为1200行已调试的 PL/I 语句（介于100万到200万个指令之间）。[10]

**10.** Corbatò, F. J., "Sensitive issues in the design of multi-use systems," lecture at the opening of the Honeywell EDP Technology Center, 1968.

该数字非常令人兴奋。与其他项目一样，MULTICS 包括控制程序和语言翻译器。和其他项目一样，它产出的是经过测试和文档化的系统编程产品。在所包括的工作类型方面，数据看上去是可以比较的。该数字是其他项目中控制程序和翻译器程序生产率的良好平均值。

但Corbato的数字是**行**/人年，不是**指令**！系统中的每个语句对应于手写代码的3至5个指令！这意味着两个重要的结论。

•对常用编程语句而言。生产率似乎是固定的。这个固定的生产率包括了编程中需要注释，并可能存在错误的情况.[11]

•如果使用合适的高级语言，编程效率可能会提高5倍。[12]

**11.** W. M. Taliaffero also reports a constant productivity of 2400 statements/year in assembler, Fortran, and Cobol. See "Modularity. The key to system growth potential," *Software*, **1**, 3 (July 1971) pp. 245–257.

**12.** E. A. Nelson's System Development Corp. Report TM-3225, *Management Handbook for the Estimation of Computer Programming Costs*, shows a 3-to-1 productivity improvement for high-level language (pp. 66–67), although his standard deviations are wide.

**#SuperMemo Reference:  
#Title: 胸有成竹（Calling the Shot）  
#Date: Imported on April 22nd, 2024  
#Parent:** [**3227**](file:///E:\Git-SuperMemobackup\SuperMemoBackup\sm18-lazy-package-1.3.1\sm18\systems\ALL%20IN%20ONE\temp\SuperMemoElementNo=(3227))**: 人月神话**