软 件 移 植

——原理和技术——

陆汝钤 韦梓楚 编著

国防工业出版社

软件移植

——原理和技术

陆汝钤　韦梓楚　编著

国防科技大学出版社

内 容 简 介

软件移植是软件工程的一个重要分支，是广大软件工作者经常遇到的一个实际问题。本书全面介绍了软件移植的各种实现技术及其原理。全书分为五篇共十五章。第一篇概论。第二篇介绍开发可移植软件的技术。第三篇介绍一项大型软件移植工程——XR 计划。第四篇阐述了移植已有软件的技术。第五篇介绍特种软件的移植，包括数学软件、操作系统及数据库。

本书取材全面，内容新颖。从系统软件到应用软件，从高级语言到汇编语言，对各类软件的移植技术均有所阐述，并附以大量实例。本书既可供从事计算机软件研制、开发和应用的专业技术人员参考，也可供高等院校计算机专业的师生阅读。

软 件 移 植

——原理和技术

陆汝钤　韦梓楚　编著

*

国防工业出版社　出版、发行

（北京市海淀区紫竹院南路 23 号）

（邮政编码　100044）

新华书店经售

国防工业出版社印刷厂印装

*

787×1092 1/16　印张 17¼　400 千字

1991 年 4 月第一版　1991 年 4 月第一次印刷　印数：0,001—3,000 册

ISBN　7-118-00330-1/TP・40　**定价：13.00 元**

谨以本书献给所有关心、支持过 XR 计划，
尤其是为它作出了贡献的人们

——代序

前　言

从 1975 年到 1981 年，本书作者和国内计算机界的一些同仁，有感于国产计算机上软件的匮乏，想通过软件移植的途径来迅速改变这一状况，从而发起并实行了一个以用高级语言编写各类软件并使软件移植机械化为中心内容的计划，即系列软件计划，简称 XR 计划。在短短几年内，XR 计划得到了全国 20 多所高校和研究机构的计算机工作者的响应和支持，形成了一个较大的科研协作集体，通过七年的努力，我们基本上实现了自己的技术目标，并荣获中国科学院重大科研成果一等奖。当然，这首先要归功于广大参研人员的辛勤努力，以及中国科学院和电子工业部计算机工业总局的关怀与支持。

在第一届全国软件工程研讨会期间，国防工业出版社的同志提议作者以这项成果为中心写一本软件移植的书。这项提议不仅促使作者从技术上对 XR 计划作一全面的总结，也促使了作者去进行进一步的收集、阅读、分析国内外有关软件移植的文献和技术。整理上述文献的过程实际上是作者重新学习软件移植技术的过程。我们发现，软件移植的天地比我们原来了解的要广阔得多。为了比较全面地把这些技术介绍给读者，我们决定，紧缩叙述 XR 计划的篇幅，而把大部分章节用于描述软件移植的不同技术流派。

软件移植是一个十分含混的概念，它的定义本身就不清楚。许多人都说要利用软件移植技术，但他们的真正目的往往是很不一样的，因而他们选用的移植技术也就不一样。对于这些问题，我们在第一章中作了较详细的分析，读者可以把这一章看成是阅读本书的指南。

本书共分五篇。第二篇和第四篇是全书的核心，分别叙述了开发可移植软件的技术和移植已有软件的技术。这是软件移植技术中最主要的两个方面，过去人们往往不大注意区分。XR 计划研究的就是开发可移植软件的技术，因此第三篇是第二篇的一个实例。此外，第二篇到第四篇以编译程序的移植为中心，兼顾一些应用程序的移植。第五篇则集中讨论编译程序以外的大型软件移植，其中采用的技术分别来自第二篇和第四篇。

目前尚无一套软件移植的完整理论，作者能做的只是分类介绍各种实用技术，供读者选用。希望读者在利用本书介绍的技术时，密切注意新的成果和新的研究动向。本书书稿完成于 1984 年，1986 年作了少量增补。书中难免有错误、不当和过时之处，望读者指正。为了弥补这一不足，书后增添了少量近期文献。

周龙骧教授审阅了第十五章初稿，在此表示感谢。

目　录

第一篇　概　论

第二篇　开发可移植的软件

第三篇　系列软件计划(XR 计划)

第四篇　移植已有软件的技术

第五篇 特种软件的移植

第一篇　概　　论

第一章　什么是软件移植

1.1　软件移植的动机

我们有理由相信，在本书的读者中间，有许多人是带着问题来读的。由于带的问题不同，关心的角度也不一样。大部分读者可能是出于实用的目的，但也有一些读者是把软件移植作为一个课题来进行研究的。还有这样的读者，他们早已和软件移植打过交道，或者他们在工作中遇到过的种种问题正是因为软件移植困难而引起的，但却未曾意识到这一点。亲爱的读者，你属于其中的哪一类呢？你也许会作出这样或那样的回答，但不管你的回答是什么，我们可以有把握地说，你对软件移植的兴趣，肯定来自上述两大类动机之一，或两者兼而有之。如果你是属于第一类，那你多半是由于关心某个或某些具体软件的移植，这是一些已经研制完成并成功地运行了的软件，现在要换一个环境，有什么办法能使它们在新环境中继续成功地运行呢？如果你是属于第二类，那你关心的不是某个具体软件的移植，而是有关软件移植的一般性课题。比如，软件移植究竟有哪些办法？什么办法适合于移植什么样的软件？进一步，软件应具有什么样的结构和体系，才便于移植？再进一步，软件移植的可行性和效果究竟如何，软件移植的极限在哪里？哪些软件是不可移植的等等。你的目的可能是要从软件设计和实现的途径上进行改革，以便从根本上解决软件的可移植性问题。

1.1.1　第一类兴趣

对于第一类关心软件移植的读者，我们还可以细分出如下几种，请读者考虑自己是属于哪一种。

(1)你是否有一个或一批应用程序，这些程序已经用惯了，但由于某种原因(或者原来使用的机器出故障，一时无法用；或者计算量增大，原机器无法应付；或者机器要更新)，你必须把这批程序搬到另一台机器上去运行。

(2)由于工作需要，你很想使用某个软件，而且你确知有人已经成功地开发了这个软件，只可惜是在别的机器上搞出来的。如何能把这个软件搬到你的机器上来，为你所用呢？

(3)你所在的单位添置了一台新机器，但新机器和老机器也许是不相容的。你在老机器上开发的一批软件能否在新机器上使用呢？反过来也有这样的情况，新机器带来了不少有用的软件，那么老机器能否分享呢？

(4)你不是一个专业用户，而是专职的软件研究人员或软件工程师，可是，你总觉得你的机器上的软件太少，太贫乏，你想尽可能多地把外界的软件都搬到你的机器上来，以丰

富本单位的软件资源。有什么办法能做到这一点呢?

(5)你研制出了一种新软件,希望迅速推广,为尽可能多的人所使用;你希望你的软件能适应更多的机器,而且能尽可能容易地在不同的机器上安装运行(移植工作量少)。

上面我们列举了要求移植已有软件的五种不同情形。这五类移植,动机不一样,要求(移植软件的多少,移植工作量及移植后软件的性能)不一样,条件(对被移植软件和目标环境的熟悉程度以及机器的可用性)也不一样。表 1-1 是有关条件和要求的一个对比。搞清楚这个是很重要的,因为它将决定我们采用什么软件移植技术。在"要求"一栏中,本应有移植后软件的"效率"一项,但由于几类软件移植对效率的要求都差不多(对效率降低的能忍受程度主要与所解决的问题本身有关,而不是与软件移植种类有关),故未予列入。

表 1-1 五种软件移植的对比

	特 征	第一种	第二种	第三种	第四种	第五种
条件	程 序	熟 悉	不熟悉	①	不可知	熟 悉
	环 境	不熟悉	熟 悉	①	熟 悉	不可知
	宿主机	无 用	不可用	可 用	不可用	可 用
	目标机	不可多用	可 用	可 用	可 用	不可用
要求	数 量	个 别	个 别	批 量	批 量	个 别
	功 能	不能变	可伸缩	①	可伸缩	基本不变
	工作量	不能太多,时间紧迫	小于自行研制的工作量	一般不集中	无限制,可分散在长时期中	由发放者和接受者分担。前者负责设计,后者负责安装

① 视情况而定,往往兼有第一种和第二种的规定。

表中所列的"环境"指的是目标机环境。"数量"指的是被移植软件的数量。注意,软件有大小繁简之分,因此数量少不等于移植工作量小。此外,"不可知"指的是移植者所要对付的软件或机器没有限制,因而无法完全预测并制定确切的计划。

1.1.2 第二类兴趣

至于第二类读者,感兴趣的范围可能是很广的,我们在这里举出几种可能性。

(1)你想要研究在一组不同型号的计算机(至少两种)之间进行软件移植的可能性。说得窄一点,是针对一组特定的机器寻求最好的移植方法,说得宽一点,是寻求一种方法,它可以根据不同的机器组合来确定比较合适的移植方法。

(2)你的目标没有(1)中所说的那样大,只想研究性能十分相近的一组机器之间的软件可移植性。最常见的例子是系列机之间软件的兼容性,这是生产厂家十分关心的问题。

(3)你的兴趣在于研究某一类软件的可移植性。按功能划分,本书涉及的软件可分为五大类,应用程序、编译程序、数学软件、操作系统及数据库。其中数学软件是一组应用程序的集合,在移植上有其特殊性。

(4)你要研究用某一种语言书写的软件的可移植性。例如,用 FORTRAN 写的应用程序的可移植性,用 ADA 写的系统程序的可移植性等等。也许你关心的不是 FORTRAN、ADA 这样的高级语言,而是汇编语言和机器语言等低级语言写的程序的可移植性。

(5)你的兴趣是对移植工具的研究。移植工具是软件工具库的一个组成部分,与其它领域的软件工具既有公共部分也有不同部分。由丰富而强有力的移植工具构成的集合可以成为一个很好的移植环境。

(6)你研究的是更一般的软件移植技术问题,它包含(5)中的软件工具,但不仅如此,作者认为,有一个简单的描述公式:

$$\text{软件移植技术} = \text{软件移植方法学} + \text{软件移植工具}$$

至于什么是软件移植方法学,我们将在下一小节中解释。

(7)最后,你研究的是软件移植的某些原理性的问题。例如上面提到的软件移植的可行性、效率、极限等等。

在下一节中,我们将针对不同的课题向读者提示一些可供参考的软件移植技术(包括软件移植方法学和软件工具),并介绍读者去参阅本书的有关章节。应该说明,上述分类并不是绝对的,有许多技术是公共的,对不同的问题往往只要变通一下即可采用。此外,由于条件所限,本书重点在于叙述具体的移植技术,对于原理性的问题只能在适当地方略为提及,不可能详细讨论。

1.2 软件移植的技术

在本节中,我们将向读者提一些建议,这样做对于那些带着问题来阅读本书的读者尤为有益。因为你会迅速地确定本书能给你什么帮助,在什么问题上和什么界线之外本书只能是爱莫能助,如果它对你还有所帮助的话,那末,这种信息应该到什么地方去寻找。

1.2.1 第一类技术

这类技术适合于第一大类读者。

如果你是属于第一种情况(有一批熟悉的程序要换个环境使用),那么,首先应看看你的程序是用什么语言写的。若是用机器指令写的,那么,我们建议你放弃移植的愿望,因为在表 1-1 所列的条件中没有什么好的办法。如果这个程序很重要,你非用它不可,并且时间上还允许,那就可选用目标机上你最熟悉的一种高级语言来重写它,这是唯一的现实途径。如果你的程序是用高级语言写的,那么,就应了解一下目标机上有无此高级语言的编译程序。若无,则同样要放弃移植的愿望,或改写为目标机上能使用的高级语言的程序。如果被移植程序所用的高级语言在目标机上也有,则主要问题是两个高级语言版本之间的兼容问题,请参阅本书第二章(当然也可用下面将提到的交叉编译)。

如果你是属于第二种情况,则你的机动余地就大多了。首先,移植用机器语言写的程序不再是不可考虑的了,可以采用直接翻译的方法(见 10.4)。如果目标机上有可编程的微程序,则可以采用硬件仿真方法(见 10.5)。你也可以使用软件技术,首先把被移植的机器语言程序提高一个抽象级别,即反汇编方法(见 12.1)和反编译方法(见 12.2)。这不仅有利于翻译成目标机的指令,也有利于分析和理解从外界引进的程序。例如,对于以二进

制形式记录在介质上的软件，就可以采用反汇编方法来为你解忧。

若被移植程序是用高级语言写的，则另有一套技术可供采用。你的目标机上有该高级语言的编译程序吗？它与宿主机上的同类编译程序能兼容吗？若回答均为“是”，当然没有问题。如果目标机上无此高级语言的编译程序，则可考虑间接翻译的方法，设法另找一台能接受该语言的机器（由于在目前讨论的情形下宿主机一般不可用，因此这台机器是第三方。但也有宿主机可用的情况，这两种情况我们在这里均统称宿主机），该台机器上有一个翻译程序，能把此高级语言翻译成目标机可接受的形式，或者是另一种高级语言，或者就是目标机的机器指令。前一种方法叫拖带，后一种方法叫交叉编译，它是拖带的特殊情形（两者均见第十一章）。拖带可以象接力赛跑似地经过多次转译，其中一部分可以在宿主机上，另一部分在目标机上。若目标机上有比较好用的系统程序设计语言，而想要移植的程序又比较重要，则也可考虑用此系统程序设计语言专门写一个程序把被移植程序所用的高级语言翻成另一种能被目标机接受的高级语言，使拖带完全在目标机上进行（见第三章和第十一章）。拖带中的某些工作可以用人工来进行。当被移植的程序数量不是太多时，与其专门写一个翻译程序，还不如自己用手改写（但应改写为目标机上能运行的高级语言，切勿改写成机器指令）。

如果目标机上有被移植程序所用高级语言的编译程序，则主要问题是该编译程序在宿主机和目标机上两个版本的兼容问题。对于程序中不能兼容的地方，既可以用手改，也可以用某些机械的方法改（见 12.4）。但仍如上面所述，凡采用机械化方法的，只有被移植程序的批量较大时才是合算的（以研究为目的的读者不在此列）。

不管用哪一种方法移植别人的程序，都可能会遇到“卡壳”的情况。此时往往要把部分程序结构弄清楚才知道应该修改什么地方。当程序难以读懂时，有些机械的方法可以帮助你，即反编码技术（见 12.3）和程序分析技术（见 12.5）。

第一大类的第三种读者，可参用第一、第二和第四种读者可用的技术，因为现在还没有在这种情况下特别适用的技术。此外，第二大类中的第一和第二两种读者所选用的技术对第一大类的第三种读者也有一定的参考价值。

至于第四种读者，则你的选择余地更大。首先，凡第二种读者可以采用的技术都可以采用。除此之外，如果需要大量移植用某种高级语言写的应用程序，则不妨直接建立该高级语言的编译程序。但这个编译程序要尽量符合该语言的通用标准，切勿任意简化，同时，应注意我们在第二章中对编译程序功能的标准化问题所作的分析。如果这个编译程序可由外界移植过来，那最好（请见第三章）。至于编译程序的移植，在目前阶段，第四章中有关编译程序的编译程序的介绍，可以作为移植和开发编译程序的一种技术加以考虑，因为这种方法现在还处于研究阶段，并不十分成熟。第十章，直接转换法中有关高级语言直接转换的诸节，目前也是研究性的，未见有生产性的应用。

对于第五种读者，有一种比较成熟的方法，即抽象机方法，该方法在第六章中有详尽的叙述。发放编译程序的人负责设计一个抽象机，并提供由源语言到抽象机的编译程序，接受者的任务是实现抽象机，并由此而实现整个编译程序。抽象机的实现方法有编译、解释、仿真和宏展开等，相关的内容包含在第三、五、八、九、十以及十一章中。

1.2.2 第二类技术

这一类技术适合于第二大类读者。

对于其中的第一种读者，抽象机方法也许是最好的方法。本书作者曾与许多同志一起，从事过一项称为 XR 计划的具有一定规模的软件移植实践（见第八和第九章），可供读者参阅。

对于第二种读者，除了享有第一种读者的共性以外，还可以考虑一些特殊的方法。例如，系列机内部的上下兼容问题往往用仿真的办法来解决。由于这个内容已经进入硬件领域，超出了本书的范围，因此只在 10.5 中略有提及。另外，宏加工技术不仅可作为抽象机的应用而为这一种读者服务，它作为预处理技术对程序在机种间的兼容性也很有用（见 5.1）。

研究某一类软件可移植性的读者，请参见有关章节。其中第五篇各章分别讨论了数学软件、操作系统和数据库的移植。但请注意，第五篇不是本书的重点，本书主要探讨应用程序和编译程序的移植，尤其是后者。第三章的标题并不表示只有这一章才讨论编译程序的移植。实际上，许多章节都涉及编译程序的移植，它是我们的主要研究对象。

第四种读者中研究高级语言程序可移植性者，可能首先关心常用语言的标准化问题，请见第二章。在那里我们还评述了高级语言程序在国内的可移植情况。但是，你也可能注意到了，一个语言的设计对可移植性会有不小的影响。关于这个问题，请你读一读第七章语言分层和语言系列。在那里我们通过一些实例提出了可能适合于软件移植的某些语言分层方案。

关于机器语言程序的可移植性问题，请参看 10.4、11.1 和 11.2。这里所说的机器语言，也包括汇编语言在内。

第五种读者的兴趣在于移植工具。本书中提到的移植工具散布在各章中。首先，高级语言及其编译程序是最重要的移植工具，它几乎是一切移植的基础。第二章和第三章分别讨论了用来书写应用程序和系统程序的高级语言，对编写可移植的数学软件、操作系统和数据库有关的语言请分别见十三、十四和十五章。重要性居第二位的移植工具是中间语言和抽象机（见第六章），占第三位的是宏加工系统（第五章）。其余的移植工具有编译程序的编译程序（第四章）、反汇编程序（见 11.1）、反编译程序（见 11.2）、指令模拟程序（见 10.5）、框图生成程序（见 11.3）、程序变换系统（见 7.3 和 11.4）和程序分析工具（见 11.5）等等。这当然不可能是一个完备的清单，我们希望读者能创造出新的，更强有力的移植工具。

对于第六种读者，我们要说，方法学和工具是很难完全分开的。方法学是一种指导思想，它往往是工具的影子和灵魂，是它指导着人们去设计、研制和使用工具。本书涉及的方法学内容有：保持与机器无关的高位势（使用高级语言，编译程序的编译程序可以看作是一种非常高级语言）；提高抽象化级别，以跨越环境壁垒（第十二章）；推迟约束时间（宏加工技术和抽象机）；使用间接和借道的方法（拖带）；使用积累和渐进的方法（自展，语言的分层结构）；以及从根本上控制软件中不可移植成分的产生和蔓延（各类标准化方案和技术）。

至于第七种读者关心的问题不是本书重点讨论的对象，除了散列各章中的有关软件

移植效率等问题的讨论(第八章中有较集中的讨论)外,只在本章的剩余部分(见 1.3)研究一下有关可移植性的各种定义。

1.3 软件可移植性的定义

1.3.1 几种可能的定义

当今国际软件界对什么是可移植性有多种定义方式,其中有些经常被引用,似乎已被公认。但据我们看来,这些定义都不能令人满意,它们都含有一些无法自圆其说的矛盾之处,或者干脆就是不现实的。常在文献中出现的定义有下面几种:

定义一:如果一个软件能不加任何改变地由甲机搬到乙机使用,则此软件称为可移植的(常称 portable)。

定义二:如果一个软件能通过机械的转换(由计算机自动进行)由甲机搬到乙机使用,则此软件称为可移植的(常称 transportable)。

定义三:如果一个软件由甲机搬到乙机使用,需要移植工作者进行的修改量大大小于在乙机上重新建立该软件所需的工作量,则此软件称为可移植的。常用公式表示如下:

$$\text{可移植性} = 1 - \frac{\text{移植时的修改工作量}}{\text{重新建立一个软件的工作量}}$$

由此公式可知,当移植的修改量很小时,可移植性接近于 1,当移植修改量接近于重新建立一个软件的工作量时,可移植性接近于 0。

我们暂时不对这三种流行的可移植性定义作出评论,但是我们可以分析一下不同类型的读者对这些定义的倾向性。由于第二大类读者研究的是一般领域,因此,我们只能对第一大类的读者作一些分析。

如果你是第一种读者,即急于把自己的程序拿到别的机器上去运行的用户,那你大概坚决支持第一种定义。这是因为你希望不对程序作任何修改即能搬到新机器上运行。不仅如此,你还要求过去能算的题目搬到新机器上以后仍然全部能算,并且结果完全一样(注意,这个要求已超出了上述定义一的范围,请见下面的分析)。

对于第二种读者,即急于把别人的某个程序搬来使用的用户,则会选择第三种可移植性定义。因为如果移植不成,你就得自己动手开发程序,只要用移植方案比自己动手合算,你就会愿意移植。

第三种读者的想法与上面两种都不大合拍,作的选择也不会一样。这样的读者不会象第一种读者那样要求程序不需作任何改变,因他使用目标机很方便,并且有足够的时间进行修改和调试。他也不会象第二种读者那样只要代价合算就干。这一类读者不是根据移植工作量和开发工作量之比来确定可移植性,而是根据软件规模与移植工作量之比,或移植工作量的大小来权衡的,即:

$$\text{可移植性} = \frac{\text{软件规模}}{\text{移植工作量}}$$

或

$$\text{可移植性} = \frac{1}{\text{移植工作量}}$$

如果你是第四种读者，则你十之八九是倾向于第二种可移植性定义的，因为你要移植的软件在时间上和数量上都是无限的。你要的是“多”，而并非专对某个软件有特殊兴趣，你也不会愿意采取逐个应付的办法。在这种情况下，机械化转换是合理的解决办法。

第五种读者对可移植性的定义不会和前四种一样。这里有一个根本的区别。前四种读者都是被移植软件的接收者或主要是接收者，而第五种读者却是被移植软件的发放者，他很难去逐个测量实际的移植工作量，只能从发放时的条件来判断，而最主要的条件就是用高级语言编写。我们不妨代这样的读者拟一个公式：

$$\text{可移植性} = \text{该软件所用书写语言的普及性} = \frac{\text{配有该书写语言的机器台数}}{\text{总共机器台数}}$$

这里的书写语言不仅适用于高级语言，也适用于机器语言。如果只有一种机器，指令程序的可移植性当然是 1。

1.3.2 分析和批评

上述分析的目的是为了说明：各种可移植性定义皆有其自己的背景，而在某种特定背景支配下给出的定义自然不可能是非常全面的。现在，我们来找找这些可移植性定义（已扩大到五种）的弊病。

第一种可移植性定义实际上是一个 0，1 函数 P：

$$P(S)=\begin{cases}1 & \text{移植时不需修改}\\ 0 & \text{移植时需要修改}\end{cases}$$

但实际上，不存在移到任何机器上都能不加修改地使用的软件，因此

$$P(S)\equiv 0,\quad \text{对所有 } S$$

这就已经显示了这种定义的不合理性。但我们还要指出：即便把“不作修改”的要求放松，第一种读者要求“原有的题目搬过去以后仍然能算，结果全部一样”也是不现实的。瑞典的达尔斯特兰德（Dahlstrand）出了一个简单的题目：400 万×400 万。他让四台不同的计算机分别计算，竟得出了四种不同的结果！

第一台：正确，答案是 16 万亿。

第二台：算出 16 万亿 mod(2^{35}或 2^{36})，计算错！

第三台：运行时错误，溢出！

第四台：编译时错误，告以运算数太大！

为什么会出现这种情况呢？因为每台计算机能表示的整数、实数等其大小都有一定的限制，且互相差异可以很大。正是由于这一点，计算机语言的文本一般都不规定它能计算的数的大小界限，以便适用于各种机器。但也正由于这一点，使得用高级语言写出的程序仍可因数据范围问题而不能顺利移植。

计算机的不同运算精度也是妨碍移植的一个潜在因素。例如用数值方法求近似解时，常会用到这样的语句：

```
while abs (x−y) gr 10 * * (−8) do
...
```

如果目标机的精度达不到 10^{-8}，则这个程序就根本不会停机！

解决的办法可能有如下几种：

(1)降低要求，用手工修改，例如改成

```
while abs(x−y) gr 10 * * (−6) do
...
```

这就已经不是完整地移植原程序的功能了。

(2)同样是降低要求，但采用一种与机器无关的方法，以避免每次手改，例如：

```
while abs(x−y) gr min real do
...
```

这里 min real 代表目标机上的最小正实数，可以因机型而异。

(3)采用软件技术，如双精度(FORTRAN)，多字长(ALGOL 68)。例如用 ALGOL 68 可写成

```
while abs (x−y)>long small real do
...
```

甚至在某些语言(公式处理语言)中可以用无限精度(只要内存放得下)，但是用软方法来对被移植软件进行全面功能模拟在时间和空间上的代价都是巨大的。图 1-1 是功能移植和效率影响之间的关系(示意)。要求功能移植很少时效率接近于 1，全功能移植时效率接近于 0。图中的虚线表示功能移植的合理界线，如果一定要过这条线，效率就会急剧下降。

我们的结论是：不但“无修改移植”是不现实的，“全功能移植”也是不现实的。

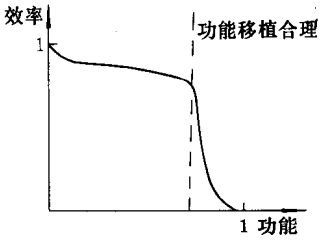

图 1-1　功能移植对效率的影响

第二种读者支持的可移植性定义(即移植工作量远小于开发工作量时称此软件为可移植的)是目前在软件界最流行的定义，但这种定义的不合理性是显而易见的。

首先，移植工作量的大小取决于被移植软件对目标机的适应程度。按照这种定义方法，由于一个软件对不同目标机的适应程度不一样，它的可移植性就随目标机而异，这样就没有一个统一的度量来衡量它的可移植性。因之，可移植性这个概念就不成为一个软件的固有性质了。

其次，即使是同一种型号的机器，提供的硬件和软件支持也是千差万别的。存储容量有大有小，外部设备的工作方式不一，使用的编译程序和操作系统也可以各异。由于这些因素都能影响移植，使得上述可移植性概念不仅依赖于机器种类，还依赖于具体机器。而且，开发工作量也不是固定不变的，随着目标机上配置新的软件工具，开发工作量可能下降。尽管移植工作量没有变化，但按本定义计算出的可移植性却反倒降低了！

一般来说，软件规模越大，开发越费事，而移植工作量的增长并不一定同样地快。当软件规模变小时，开发变得相对地容易，而移植工作量和开发工作量之比是与软件规模有关的，软件规模增大时，比例变小；软件规模缩小时，比例增大(见图 1-2)。结论竟是①：软件

① 在固定目标机环境等一系列条件下。

规模越大，可移植性程度越高，软件规模越小，可移植性程度越低。于是，规模大到天文数字般的软件可移植性最好，而只有一个语句的软件是不可移植的，因为移植工作量等于开发工作量！为什么会有这样荒谬的结论呢？原来这个定义把可移植性与移植相对于开发的益处，这两个概念混为一谈了。

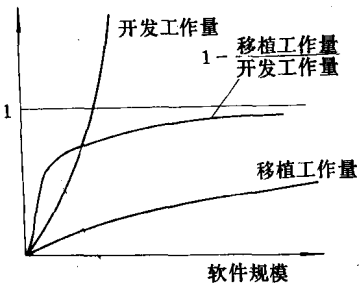

图 1-2　软件规模对开发工作量和移植工作量的影响

第三种读者对可移植性所给出的第一种定义也是基于由软件移植得到的益处。不过不是与开发工作量相比，而是与被移植软件的规模相比。它与刚才讨论过的定义(即定义三)有内在关系。令定义三的可移植度量为 P，这里定义的可移植性度量为 P'，又假设软件规模和开发工作量成线性关系，并假设线性因子为 α，则：

$$\text{软件规模}=\alpha\times\text{开发工作量}$$

则两种可移植性度量有如下关系：

$$P'=\frac{\alpha}{1-P}$$

事实上，本定义确有和定义三类似的弊病。首先是它依赖于具体的机器，顶多只能表示被移植软件对特定机器的可移植性。其次，从本定义也可得出类似的结论：软件规模越大，则可移植性越好。但也有一点不一样，即当软件规模变小时其可移植性度量不一定很小，因为此时移植工作量也小了。本定义把软件的可移植性和软件规模联系在一起也有合理的一面，我们在以后将要参考这个思想。

第三种读者给出的第二种定义具有和第一种定义类似的缺点(依赖具体机器)，且完全抛弃了软件规模和开发工作量的概念，虽然不会作“软件越大，越容易移植”的错误判断，却也忽视了当软件规模变大时，应当允许移植工作量也适当增大这一合理要求。根据这个定义，可移植的界线是很难划分的。划高了，大型软件都将列入不可移植的范围；划低了，中小型的机器指令程序也都是可移植的了。所以，它供实际工作参考尚可，作科学的定义则不行。

第四种读者的可移植性定义看来比较符合一般人的想象。人们希望的是由计算机来代替人的劳动，软件移植当然也不例外。但仔细推敲起来，它在理论上和实践上都不是无懈可击的。首先，实践经验告诉我们，软件移植不可能百分之百地靠机械进行，如果仅仅因为人参加了百分之一的劳动，就否定软件的可移植性，似乎不够公平。其次，担负机械移植任务的软件工具本身需要开发，这个软件工具的开发工作量是否算在软件移植工作量之内呢？比如研制 COBOL 编译程序的工作量是否算在移植 COBOL 应用程序的工作量之内呢①。实际上，单为移植一个软件而建立机械翻译工具的情形是很少见的，机械翻译一般都是针对一类软件，例如用某种高级语言写的应用程序。因此，本定义用来描述一类软件的可移植性似乎更为合适。

最后谈谈第五种读者的可移植性定义。这个定义给人以武断的印象。为什么用高级

① 即有了编译程序，移植应用程序时还是有许多事情要做的，见第三章。

语言写就一定是可移植的呢?如果目标机上没有该高级语言的编译程序,可移植性又如何保证呢?当然更不用说此高级语言是发放软件者的独创、尚未普及这种极端情形了。其次,有了编译程序并不等于万事大吉,还有编译程序是否符合语言标准的问题,以及编译程序对语言功能的自然限制问题。所谓自然限制,是说编译程序实际上不可能无限制地实现语言的一切功能,除了数的大小之外①,还有过程的参数个数、表达式括号深度、分程序嵌套深度、条件语句和循环语句的嵌套深度、标识符个数、数组长度等等②,在每个系统中都是有实际限制的。除此之外,我们还必须注意到高级语言和操作系统的接口等隐藏的与机器有关的因素。鉴于上述理由,单凭高级语言这一条来确定软件的可移植性是不够的。

这种定义也有其合理之处。它是本节讨论的各种定义中唯一的一种不涉及具体机器,只依据软件本身特性来确定其可移植性的定义。而且,用高级语言编写确实也是目前提高软件可移植程度的最常用、最有效的手段。看来,它缺少的是概念上的定量化,不能作为可移植性的量度。

概括起来,所有这些可移植性定义不外乎基于如下三种出发点:

(1)第一种出发点。可移植性应该是软件本身的一种属性,不应该与目标机的特性有关。这种观点在理论上最能站得住脚,可惜它在现实中经常碰钉子。因为按照这种观点,即使是一个最简单、最标准的 FORTRAN 程序,只要世界上还存在一台没有 FORTRAN 编译程序的机器,它就是不可移植的。

(2)第二种出发点。可移植性不是软件本身的属性,而是软件相对于目标机的一种性质。于是得到这样的结论:有多少种目标机,就有多少种该软件的可移植性度量。上面讨论过的定义有好几种属于此类。但从这里出发不能得到作为一个单一度量的可移植性概念。

(3)第三种出发点。可移植性既不是软件本身的属性,也不是软件相对于目标机的属性,而是软件移植相对于人的属性,即人通过移植该软件得到多少益处(相对的或绝对的)。这个原则显然有很大的随意性,因为人的判断之间的差别可以很大,因此也不是一种科学的定义。

1.3.3 我们的观点

说到这里,读者可能要问:既然上面几种可移植性定义都不能令人满意,那么,比较确切的定义应该是什么呢?现在说一说我们的出发点。

(1)可移植性应该是软件本身的一种属性,是一个单一的度量,和具体的目标机无直接关系。

(2)软件可移植性也不是和目标机毫无关系,但是这种关系是一种总体的关系,对于每台具体的目标机来说,只有一种概率的联系,因为影响可移植性的因素太多了。

(3)可移植性和软件大小无关。

(4)可移植性不应追求不切实际的目标,例如"无修改移植","纯机械修改移植"或"全功能移植"。但是,总修改量的多少、机械修改量的多少,以及功能受限的多少应该从定量

① 见我们对第一种用户可移植性定义的评述。

② 参见第二章。

角度上对可移植性有影响。

(5)一个软件的可移植性可以随时间而变化。它应是各目标机上软硬件支持能力的递增(至少是非减)函数。

(6)软件的可移植性应该是简单而容易确定的,避免繁琐。

根据以上各条,我们给出如下的可移植性定义:

(1)凡可移植软件必须是用高级语言编写的;

(2)可移植性度量 P 定义为:

$$P = \alpha \cdot \frac{\sum_{i=1}^{N} \beta_i}{N}$$

其中,N 是计算机的总台数,β_i 表示第 i 台计算机配备该高级语言编译程序的情况。若第 i 台机器无此编译程序,则 $\beta_i=0$。对所有 β_i 均有 $\beta_i \leqslant 1$,$1-\beta_i$ 表示该编译程序偏离语言标准的程度。对 α 有 $0<\alpha\leqslant 1$,α 表示该软件使用非标准语言成份的程度。简而言之,$\sum_{i=1}^{N}\beta_i/N$ 是所用高级语言的普及程度,α 是软件本身偏离标准的程度。

下面我们回答几种责难。

(1)用低级语言写的软件都是不可移植的吗?实践告诉我们,移植这种软件的困难极大,它们的可移植性可以忽略不计。

(2)究竟用什么标准来划分高级语言和低级语言?若用霍尔斯代德(Halstead)的软件科学度量来计算语言的高级程度似乎太书生气。在一般人心目中,哪些语言算高级语言大致有个公论。或者,就拿我们的度量 $\sum_{i=1}^{N}\beta_i/N$ 来衡量语言的高级程度也未尝不可。这是一个充分条件,因为低级语言不可能在大量不同型号的机器上实现。它不是必要条件,因为有许多高级语言未大规模普及。为此,把我们的定义改为"普遍使用的高级语言。"似乎更妥当。

(3)是不是要统计全世界所有的计算机以后才能算出一个软件的可移植性?不!我们已经指明了这个定义的概率性质。因此,量 $\sum_{i=1}^{N}\beta_i/N$ 可通过抽样计算估出。如果你关心的只是某一范围内的可移植性,那只要作此范围内的抽样计算就可以了。

(4)α 和 β_i 如何计算?β_i 可利用一组标准试题测试。如对 FORTRAN 语言可利用美国的 NBS 标准试题,至于 α 的计算可选择在一个标准的编译程序上试验。

(5)为什么没有把间接的拖带和利用其它机器做的交叉编译等可能性考虑进去?这是为了简化可移植性的定义。对于接受被移植软件的用户来说,利用他人的机器作交叉编译通常是不方便的。因此,在该软件被交叉编译之前,不能认为对目标机是可移植的,而在交叉编译之后,所生成的目标软件才是对目标机可移植的。

(6)既然本书作者对可移植性下了这样的定义,为什么书中又讨论不符合此定义的软件移植?我们的定义只反映了目前各类软件可移植特性中的一个主流。而书中作为研究,自然要探讨软件移植的各种途径。而且"软件移植"的课题范围远远超过了"一个软件的可移植性"。