第一章　引　言

本章为引言。内容包括软件语言含义、软件语言作用、软件语言级别、软件语言发展以及本书内容与体式五节。

1.1　软件语言含义

1.1.1　语言

语言乃信息交流工具，其源可追溯至上古之“结绳记事”，然本书所论语言乃现代意义下之语言。今援引如下几种定义。

(1) Webster 字典定义(下作 W 定义)

An artificially constructed primarily formal system of signs and symbols (as symbolic logic) including rules for the formation of admissible expressions and for their transformation [基于一组记号与符号由人工构作之（基本上说）形式系统（如符号逻辑），包括合法表达式之形成规则与转换规则。][见 Webster's Third New International Dictionary, G&C Merriam Co, 1976, pp. 1270, c]

(2) Longman字典定义(下作 L 定义)

Any system of signs, movements, etc used to express meanings or feelings(任何表情达意之记号系统)。[见 Longman Contemporary English-Chinese Dictionary，现代出版社，1988 年 11 月，第 789 页。]

(3) 英汉双解计算机辞典定义（下作“双计”定义)

A set of characters, conventions and rules, that is used for conveying information. The three aspects of language are pragmatics, semantics and syntax[一种用于传递信息之字符、约定和规则的集合。语言的三个方面是语用学（笔者按，此处应为语用）、语义学（笔者按，此处应为语义）和语法。][见“英汉双解计算机辞典”，清华大学出版社，1996 年 2 月第 2 版，第 553 页。]

(4) 中国大百科全书，电子学与计算机卷定义（下称大百科定义)

语言的基础是一组记号和一组规则，根据规则由记号构成之记号串的总体就是语言。[见‘中国大百科全书,电子学与计算机’卷 1(‘程序设计语言’词条)，第 86 页，中国大百科全书出版社，1986 年 9 月。]

笔者认为，前述四种定义各有千秋，其中 W 定义基本可行，惜文字表达欠佳，且严谨不够；L 定义立论过泛，且欠严格；双计定义内容欠严谨；大百科定义相对较为合适，今以之为基础，将‘语言’定义如下：

语言是基于一组记号与一组规则、根据规则由记号构成之记号串的总体。任何语言均包括语法、语义和语用三方面。

1.1.2 软件

计算机软件（简称软件）是计算机系统中的程序及其有关文档。程序是计算任务之处理对象与处理规则的描述，文档是为了便于理解程序所需之资料说明，程序必须装入到机器内部才能工作，文档一般是给人看的，不一定装入机器。

细言之，软件具有如下三层含义：

第一，个体含义，软件是指计算机系统中各别程序及其有关文档，如特定语言之编译程序及其有关文档。

第二，整体含义，软件是指计算机系统中个体含义下软件之总体。

第三，学科含义，软件是指开发与维护前述含义下之软件所涉及的理论、原则、方法与技术。在这种含义下，宜称为软件学，一般亦称软件。

1.1.3 软件语言

软件语言是用以描述软件的语言。申言之，软件语言用以描述软件、软件开发过程中间结果，以及软件开发过程本身（当然，后二者亦可视为软件）。软件语言质量关系到软件质量(如易读性、易维性、功效性、安全性等)、软件开发功效以及软件、软件技术、软件科学之发展。它一直是计算机系统之重要组成部分，一直是计算机科学与技术之重要组成部分。

1.2 软件语言作用

1.2.1 描述作用

软件语言是软件描述工具，程序及其文档赖于语言得以表述，未经描述的程序及其文档只能存在于开发人员脑中，无法为人知晓，从而也难以臻于实用。“内容决定形式，形式影响内容”。一方面，相对软件内容说来，语言是描述工具，描述是形式，但另一方面，语言本身也有内容与形式两面。语言成分是内容，语言描述（表示）是形式，因此，语言之优劣也会影响到软件内容，例如，语言结构决定了程序结构，程序结构又影响到程序之易读性、易维性、安全性等等。文章之好坏，意境固然重要，但文笔之通畅优美亦不容忽视。它往往影响到读者情绪，影响到作品传世之久远。举凡传世之作，莫不意境高超、文笔优美。程序设计语言 ALGOL 68 就其内容而论，确有不少创新之处，惜其“语言报告”文笔晦涩，不便阅读，难于流传。反之，PASCAL “语言报告”却文笔流畅，清晰易懂，二者成一鲜明对比，类似事例比比皆是，不胜枚举，描述之重要性可见一斑。

1.2.2 交流作用

软件语言又是软件交流工具。经软件语言描述之软件方可在社会上交流。如前所述，未经语言描述之软件只存在于开发人员脑中，无法进行交流。软件开发之目的是为了实用，为了在社会上得到广泛应用，以促进物质生产、科技文化、人民生活之发展与提高。但是，就特定软件而言，经语言描述后，原则上均可交流，但交流范围之广狭，生存期之长短，除了取决于软件本身质量外，其描述之优劣亦颇重要，历史上有关机构曾竭力推广

PL1 语言,然终难如愿,原因乃在 J 语言本身。ADA语言虽经美国国防部硬性推广，且内容不断更新，能否经久而不衰，殊难逆料。由于计算机网络之盛行，由于社会信息化之发展,JAVA 语言一经推出，顿成热点，原因仍在语言本身，如能对之改进润色，更上一层楼，则用户幸甚。

1.2.3 标志作用

软件语言既是描述工具，又是交流工具，无软件语言即无软件，其地位可谓举足轻重。环顾软件发展历史，代表性语言莫不成为软件发展之重要标志。50 年代中期 FORTRAN 语言出现，显著减轻了程序人员的手工劳动，提高了程序设计功效，促进了计算机的推广应用。ALGOL 60 及其相应语言族之发展，推动了科学计算领域及其有关科技领域之发展。COBOL语言出现大大推动了数据处理领域之发展。 LISP 语言推动了表处理之发展。PASCAL 语言推动了程序设计语言教学以及科学计算领域之研究。 SIMULA 67 语言开创了对象式（面向对象）语言之先河。随后,ADA, SMALLTALK, C, C + + , EIFFEL 以及新近开发之 JAVA 等语言均为对象式（包括基于对象）语言之发展标志。另一方面，各种函数式语言与逻辑式语言促进了计算机体系结构研究以及逻辑问题研究与发展，此外，设计级、功能级、需求级语言之发展促进了软件开发、软件开发过程、计算机辅助软件工程以及软件自动化等领域之研究与发展。凡此种种，莫不表明软件语言水平是软件科学技术水平之重要标志，同时也是计算机科学技术水平之重要标志。

3 软件语言级别

软件语言可区分为需求级语言、功能级语言、设计级语言以及实现级语言。

1.3.1 需求级语言

需求级语言意指用于软件需求阶段之语言，特别是用于书写软件需求定义（或规约）之软件需求定义语言。软件需求定义是软件需求之完整描述。软件需求包括功能需求和非功能需求。功能需求从用户角度明确软件必须具备之功能；非功能需求可包括功能限制、设计限制、环境描述、数据与通信规程和项目管理等。软件需求定义主要面向用户，采用基于客观外界之描述模型，以便于用户理解。

在计算机发展早期，待解问题规模较小，一般采用自然语言书写软件需求。随着待解问题规模日益增大，其复杂度不断提高，自然语言之非形式性导致需求定义出错，纠正并非易事。70 年代起，以软件方法学为基础，着手研究需求级语言形式化，先后提出基于自顶向下途径之“结构化分析”(SA),基于自底向上途径之“问题陈述语言”(PSL),基于对象式思想之“需求建模语言”(RML)，以及以对象及其相互关系为核心、以图形化表示机制为刻画手段、基于对象式需求模型之各种需求级语言。

按照形式化程度，需求级语言可分为非形式化语言、半形式化语言以及形式化语言三类。

需求级语言研究涉及基本模型、语言结构以及语用分析等。

需求级语言是需求工程核心内容之一，其研究已取得较大进展，现有各种语言已逐步

用于软件工程实践，效果良好，今后发展将更为迅速。

1.3.2 功能级语言

功能级语言意指用十书写软件功能规约之语言。软件功能规约是软件功能精确而完整之描述。它描述软件“要做什么”，以及“只做什么”。功能规约是需求定义之功能抽象，它对需求定义中用户所需功能重组，使之尽可能表述为数学语言，以作为设计与实现依据。功能级语言又称功能规约语言。

在软件发展早期，软件功能规约主要用自然语言书写，用户易学易用。但自然语言之歧义性、模糊性和不完备性致使用以书写的功能规约难以成为设计与实现之依据，同时也难以开展软件形式化与自动化研究。为了提高软件生产率与软件产品质量，出现了形式化功能级语言（又称形式化功能规约语言）。这类语言理论严谨，便于研究规约之性质，如一致性、完备性、等价性等等。不仅避免了自然语言之歧义性、模糊性和不完备性，而且奠定了 WP和 VDM 等形式化方法之基础，使软件自动化之实现成为可能。

从形式化程度看，功能级语言可分为非形式化语言与形式化语言。

功能级语言主要涉及规约对象、规约方法以及规约性质等。

功能级语言，特别是，形式化功能规约语言研究已有较大进展，其理论基础日臻完善，方法日趋成熟，并已逐步用于软件工程实践，进一步工作包括应用于大型软件开发以及基于形式化功能规约语言之软件形式化与自动化研究。

1.3.3 设计级语言

设计级语言意指用于书写软件设计规约之语言。软件设计规约包括概要设计规约与详细设计规约，其描述对象是软件系统组织或其组成部分之内部结构。设计级语言一方面不同于刻画软件及其组成部分界面特性之功能级语言，另一方面，它并不刻画重在执行高效之软件细节，从而也有别于实现级语言（即程序设计语言）。

在计算机发展早期，软件设计人员使用图形化或半图形化设计级语言，其中夹以自然语言正文描述，以书写概要设计规约或详细设计规约。稍后，相继出现表格形式之设计级语言和设计性程序语言。目前，形式化设计级语言颇受重视，在详细设计中应用尤多。

设计级语言主要涉及规约结构、规约对象、规约方法等。

按照传统观点，设计级语言应是可扩充语言。一方面，通过扩充，可以包容数据结构与控制结构新概念，如多任务、并行处理、进程间通信、声象界面等；另一方面，通过扩充，可支持不同应用领域之特定结构。然而，近来却倾向语言具有固定结构，这是由于计算机辅助软件工程（CASE）技术促进了设计过程自动化。因此，语言中宜包括表达能力颇强之成分，以适应不断发展之应用要求，例如，用户自定义抽象数据类型、类属模块、多继承机制、并行循环、非确定等待、进程之条件激活机制等，而领域特定结构则可通过领域专家库提供。

1.3.4 实现级语言

实现级语言意指用于书写处理算法的语言，即程序设计语言（又称编程语言），相对其它级语言而言，实现级语言发展最早、相对成熟，其优劣不仅影响程序人员使用是否方便，

程序人员所写程序质量之高低，而且对实际处理功效也影响至大。今就其基本成分、语言分类，以及主要语言简述如下：

语言种类千差万别。但是，一般说来，其基本成分不外四种，即用以描述程序中所含数据之数据成分；用以描述程序中所含运算之运算成分；用以描述程序中控制构造之控制成分；以及用以刻画程序中数据传输之传输成分。

按照语言级别，有低级语言和高级语言之分。低级语言包括字位码、机器语言和汇编语言。其特点是与特定机器有关，功效高，但使用复杂、繁琐、费时、易出差错。其中字位码是计算机唯一可直接理解之语言，但由于字位码程序为一串字位，复杂、繁琐、冗长，几乎无人直接使用。机器语言是表示成数码形式之机器基本指令集，或者是操作码经符号化后之基本指令集。汇编语言是机器语言基本指令中不仅操作码已符号化，而且地址部分也符号化后之结果，或者进一步还包括宏构造。高级语言表示方法要比低级语言更接近于待解问题之表示方法，其特点是在一定程度上与具体机器无关，易学、易用、易维护。当高级语言程序翻译成相应低级语言程序时，一个高级语言程序单位一般要对应多条低级语言指令，相应编译程序所产生之目标程序往往功效较低。

按照用户要求，有过程式语言和非过程式语言之分。过程式语言主要特征是，用户可以显式指明一列可顺序执行之运算，以表述相应计算过程。例如，FORTRAN, COBOL, ALGOL60 等都是过程式语言。非过程式语言的含义是相对的，凡是用户无法显式指明表述计算过程之一列可顺序执行之运算的语言都是非过程式语言。 PROLOG 语言即其一例。

按照应用范围，有通用语言和专用语言之分。目标非单一的语言为通用语言，FORTRAN, COBOL, ALGOL60 等均是。目标单一的语言为专用语言，如 APT 等。

按照使用方式，有交互式语言和非交互式语言之分。具有反映人－机交互作用成分之语言为交互式语言，BASIC 即是。语言成分不反映人－机交互作用之语言为非交互式语言，前述 FORTRAN, COBOL, ALGOL60 以及 PASCAL 等均是。

按照成分性质，有顺序语言、并发（并行）语言和分布语言之分。只含顺序成分的语言为顺序语言，FORTRAN, COBOL 等均是。含有并发（并行）成分的语言为并发（并行）语言，并发 PASCAL, MODULA, ADA 等均是。考虑到分布计算要求的语言为分布语言，如 MODULA * 即是。

传统程序设计语言大都以冯·诺依曼式计算机为其设计背景，因而，又称为冯·诺依曼式语言，J. Backus 于 1977 年提出的函数式语言 FP 则以非冯·诺依曼式计算机为其设计背景，因而，又称为非冯·诺依曼式语言。

主要语言举例如下：

1）自动数控程序语言 APT（Automatically Programmed Tools） 第一个专用语言，用于数控机床加工，1956。

2）公式翻译程序设计语言 FORTRAN（FORmula TRANslation）：第一个广泛使用之高级语言，为广大科学和工程技术人员使用计算机创造了条件， 1956。

3）面向商业通用语言 COBOL（COmmon Business Oriented Language）：广泛使用之商用语言，1960。

4）算法语言 60 ALGOL60（ALGOrithmic Language 60）：程序设计语言由技艺转向科

学之重要标志，其特点是局部性、动态性、递归性和严谨性， 1960。

5) 表处理语言 LISP(LISt Processing)：引进函数式程序设计概念和表处理设施，在人工智能领域内广泛使用， 1960。

6) APL(A Programming Language)：一种提供很多高级运算符之语言，它可使程序人员所写程序甚为紧凑，特别是涉及到矩阵计算之程序，直到 1967 年才定义出其实现文本，1962。

7) 模拟语言 SIMULA(SIMUlation LAnguage)：一种主要用于模拟的语言，它是 ALGOL60之扩充，1966。SIMULA67 是 1967 年 SIMULA 之改进，其中引进了“对象”及“类”等概念,它是第一个对象式语言。

8) 菲利浦自动顺序计算机语言 PASCAL(Philips Automatic Sequence CALculator)：它是在 ALGOL60 基础上发展起来的重要语言，其最大特点是简明性与结构性， 1971。

9) 逻辑程序设计语言 PROLOG(PROgramming in LOGic)：一种处理逻辑问题之语言，它已广泛用于关系数据库、数理逻辑、抽象问题求解、自然语言理解等多种领域，1971。

10) SMALLTALK：一种对象式程序设计语言。自 1971 年出现后，曾有多种不同文本,其中使用最广的是 SMALLTALK80,其显著特点是利用‘对象”,以使对象式程序设计得到广泛应用，1971。

11) C：一种使用颇为广泛的程序设计语言。它原先为辅助开发 UNIX 操作系统而设计，后来广泛用于 研究、开发与教学，主要用于系统程序设计，同时也用于其它领域， 1974。

12) FP：一种无副作用、具有组合子风格的函数式程序设计语言。其特点是引用透明、便于表示递归函数、具有潜在并行性以及具有良好代数性质。由 John Backus 提出，1977。

13) ML：一种严格的函数式程序设计语言，但非纯函数式，非引用透明，且其 I/O 系统具有副作用。由 Robin Milner 提出,1975～1976。

14) ADA：一种现代模块化语言，它属于 ALGOL-PASCAL 语言族，但有较大变动。其主要特征是强类型化和模块化，便于实现分别编译，提供类属设施与异常处理，适于嵌入式应用，1979。

15) MIRANDA：一种非严格的纯函数式程序设计语言，具有惰性计值和多态强类型特征，应用较为广泛，已有商业产品。由 David Turner 提出，1985～1986。

16) C++：一种使用广泛之对象式通用程序设计语言，它具有 SIMULA 语言之程序组织设施以及 C 语言之灵活性与功效性，1986。

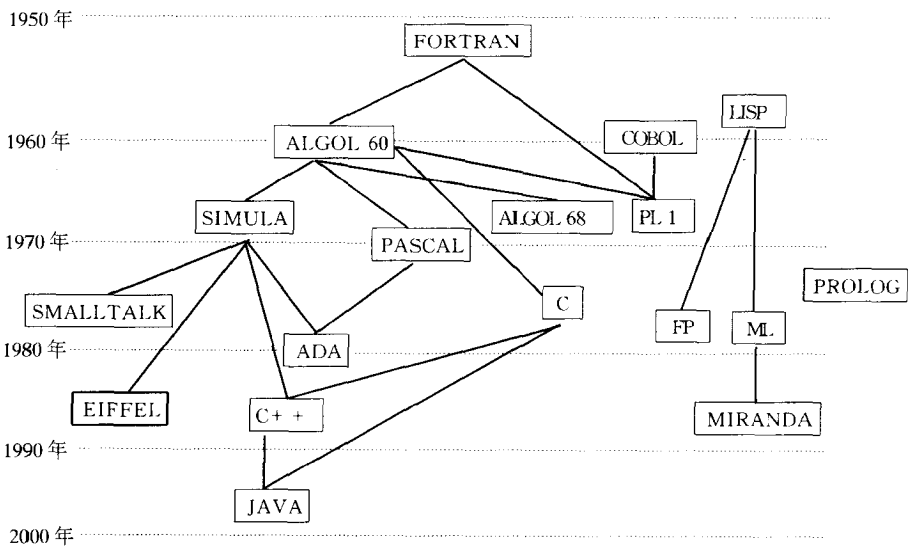
17) EIFFEL：一种对象式通用程序设计语言，该语言严格区分静态类和动态对象，类是唯一程序构造单位，并支持多继承，1987。

18) JAVA：一种适于网络的程序设计语言。其特点是简明性、对象式、分布性、健壮性、安全性、多平台运作、易移植性、多线程等，1994。

除了上面所举语言外，还有一些通用语言，特别是 BASIC, PL/1, SNOBOL, ALGOL68等。BASIC 虽然简单、易学，使用广泛，但其中并无新概念，而且并非第一个交互式语言。PL/1 之设计思想源于 JOVIAL,其功能源于 FORTRAN, COBOL, ALGOL60, 具有中断和表处理等设施。 SNOBOL 是一种好语言，对 COMIT 中若干概念做了明显改进。ALGOL68 在语言成分和描述方法方面虽有所创新，但应用尚不广泛。

表 1.1 中列出若干重要语言之衍生关系表。

表 1.1 重要语言衍生关系表



4 软件语言发展

软件语言已出现四十余年，其发展特征可归结如下，第一，就其级别而论，由低抽象级语言向高抽象级语言发展；第二，由顺序语言向并发（并行）语言发展；第三，由单机语言向网络语言发展。

1.4.1 低抽象级到高抽象级

软件语言由低抽象级语言逐步发展为高抽象级语言，亦即，由实现级、设计级、功能级向需求级语言发展。

在计算机发展早期，为了在计算机上解题，需先将待解问题提炼成数学问题，寻求合适解法与算法，用相应语言表述算法，此即为计 算机程序，简称程序。这里语言乃实现级语言，即用于直接陈述实现算法之语言。而在实现级语言内部，又是由低级语言发展到高级语言。初期，程序人员总是用低级语言（特别是机器语言）编写程序，但很快发现其弊端，使用繁琐、费时、低效、易出差错。1951 年瑞士学者 N. Rutishauser 感于低级语言使用欠便之症结在于，它与数学语言距离较大，程序人员编写程序需涉及与机器有关细节，因而，他考虑设计一种接近于数学语言之语言 L(即所谓高级语言)，但计算机并不认识 L，还需有一种机器语言程序 P,其作用是将任何用 L 书写之程序自动转换成相应机器语言程序,这样,程序人员只需用 L 编写程序即可，程序设计之功效便可显著提高。由于种种原因，直到 1956 年，世界上才出现第一个实用高级语言 FORTRAN。此后，实现级语言之发展主要是高级语言。1968 年出现软件工程，将软件开发视为工程性任务，软件开发过

程区分为需求分析、功能规约、设计、实现等阶段，相继出现了设计级、功能级以及需求级软件语言。如前所述，在各级语言内部，一般遵循非形式化、半形式化、形式化途径发展。当然，迄今也有采用非形式化、半形式化与形式化三种表示刻画同一系统之例。此三种表示之利弊在 1.3 节中业已提及，今不赘述。语言级别不同，情形也各异。对需求级语言，如考虑需求定义至功能规约转换，则形式化程度越高越佳。当然，对通用系统而言，这类转换理论上不可能完全自动进行，必须借助人 –机交互；如不考虑这种转换，而将需求定义只视为用户与软件开发人员间契约之基础，则三者兼用亦无妨，但需避免需求定义之歧义性与不一致性，尽可能提供相应检查设施。对功能级语言，则宜形式化，一则因功能规约乃衡量程序正确性之基础，二则如考虑功能规约至设计规约之自动转换，则必须要求功能规约形式化。此外，设计级语言之发展无疑走形式化道路。无形式化则无自动化，无自动化则软件开发功效便难以从本质上提高。

1.4.2 顺序语言到并发(并行)语言

早期，用户在计算机上待解之问题均为顺序性问题，亦即，问题解法所涉计算只是顺序计算，计算按步进行，各步之间无时间交迭，相应程序称为顺序程序，只含顺序成分之语言称为顺序语言，FORTRAN, COBOL, ALGOL60, PASCAL 等均是。随着计算机日益推广使用，为了提高计算机系统性能，出现了“并发”概念，其本义是多个作业运行时间的夹插(即宏观上并行，微观上串行)或交迭(真正并行)。在单处理机系统中，并发执行意指多个作业夹插执行，而在多处理机系统中，并发执行才可能真正反映多个作业运行时间之交迭。含有并发（并行）成分的语言称为并发（并行）语言，它可用于书写并发(并行）程序。并发（并行）程序提高了系统性能，促进了各种类型与不同程度之并行性开发与研究，促进了多处理机计算机系统体系结构的研究，然而，并发（并行）程序却引起如下问题。第一，非确定性。和顺序程序不同 (顺序程序基本上是确定性的，亦即，可预见其各个执行步之执行顺序，但其中也有一些语言构造导致非确定性。例如，考察无序命令组 ‘C_1, C_2’，其中一个子命令可更新另一子命令所含变量，因而，该命令组之性态便和子命令执行顺序有关)，并发程序却可以是高度非确定性的，其组成部分之执行顺序不可预见。第二，速度相关性。顺序程序和速度无关，乃因为其正确性并不取决于执行速度。然而，并发程序却一般和速度有关，因为其结果可能取决于其各组成部分之相对执行速度。第三，死锁。死锁是指一组进程由于彼此对资源的不合适要求而导致大家都无法前进之现象。当且仅当如下诸条件均成立时，便出现死锁，这些条件是，互斥、等待与拥有、无预先抢占以及循环等待。已出现各种防止死锁的方法。第四，饿死。调度是指对各个进程分配资源，以实现其各自目标，合理调度是指可保证需要资源之进程不致由于其它进程之需要而永远得不到所需资源。一进程由于不合理调度而导致永远不能运行之现象称为饿死。据此，保证调度合理便可避免出现饿死现象。

设计并发（并行)语言一般途径有二，一是在顺序语言中添加并发(并行)成分；二是重新设计并发（并行）语言。至于何种途径为优，不能一概而论，需视具体情形而定。

1.4.3 单机语言到网络语言

计算机系统由单机系统发展到多机系统和计算机网络。在一定意义下，软件语言之

发展亦如此。

理论上说，软件语言中只有实现级语言之低级语言才和具体机器有关，其余语言均和具体机器无关。申言之，实现级语言中的高级语言虽和具体机器无关，但其设计背景仍基于一类机器,特别是,冯·诺依曼体系结构计算机。设计级语言接近于实现级语言，情形类似。功能级与需求级语言原则上所描述的是系统所应具备之功能（当然，需求级语言中尚有描述非功能需求之成分）,描述系统"做什么",而不涉及系统"如何做",因而,应和机器无关。在这种意义下，实现级语言中低级语言除外的所有单机语言均适用于多机系统与计算机网络，然而，要使语言能实际起作用，还需考虑其实现、考虑语言实现环境、软件开发与运作平台。早在计算机问世以前，1936年A.M.Turing即已证明:"任何计算机均可起另一计算机之作用"。因此，上述实现环境、开发与运作平台等理论上亦无问题，具体而论，则要涉及实现策略、实现算法，特别要考虑到相应解释程序之功效。

单机语言如只考虑用于描述顺序程序，则无需考虑并发成分；否则，需设置并发成分。多机语言原则上应考虑并发成分。计算机网络使人类最大受益是，网上资源共享（如共享特定计算机、共享特定计算机软件，以及共享其它价格昂贵或唯一因而需共享之设备）、数据共享（如对局域或远程用户提供取接唯一之数据库）以及网上各结点间之通信与数据交换。为此，软件语言需能适用于多种开发与运作平台，乃因一般网络可为异构型之故。如前所述，这在实现上并无理论问题，但欲使其现实可行，则需考虑相应解释程序之功效，其所占存储空间与解释速度需能为一般用户接受。以往由于网络应用尚未普遍，软件语言设计者对此未多注意，近来，网络应用风起云涌，已成燎原之势。JAVA语言设计者有鉴于此，开发了规模不大而又具基本功能的解释程序，初步解决了存储空间矛盾，惜其功效欠佳,近闻其2.0版之功效约可提高一个数量级，诚能如是，则用户幸甚。当然JAVA语言中尚有其它有待改进之成分。尽管如此，它仍不失为一种适用于计算机网络之语言。

1.5 本书内容与体式

1.5.1 取材以作者工作为基础

本书作者致力软件语言及其实现研究历有年所，涉及实现级、设计级、功能级、需求级等各级语言，风范涉及命令式、函数式、逻辑式以及对象式。迄今已研制出16个软件系统，同时在形式语义、类比推理等方面亦有成果。

早在60年代初，感于机器语言程序编写工作之繁琐、费时、易出差错，作者之一曾率研制组在J-501计算机上开发出国内首例ALGOL编译系统，交付使用后，又相继开发出103等机ALGOL编译系统。此诸工作均曾在全国计算机会议上介绍交流。

70年代，作者感于当时书写系统程序之语言多为低级语言，程序人员将系统程序之开发视为畏途，曾率研制组在655机上开发出自编译系统NDHD,用于编写编译程序,相应生产率得以显著提高。国内首届计算机操作系统会议后，又率研制组开发出主要旨在书写系统程序（特别是，书写计算机操作系统等并发程序）之通用系统程序设计语言XCY（后发展为XCY语言族）,并在DJS 200系列计算机上实现，书写出DJS-240操作系统全部，以及若干编译程序。该项工作曾于1980年10月在日本东京召开的第八届世界计算机大会(即IFIP'80)上专题介绍。

80 年代，作者一面研究新型程序设计语言，如函数式语言、逻辑式语言等，对函数式语言 FP 分别在冯·诺依曼计算机与数据驱动计算机模型上实现，还自行设计逻辑式与函数式结合之语言 KLND，以作为并行推理系统之核心语言；一面致力研究设计级与功能级语言，设计出 GSPEC, FGSPEC 等，并相应研制出多个软件自动化系统。

90 年代，一面研究对象式语言及其形式语义，兼及类比推理；一面研究需求级语言之设计与实现。设计出三个需求定义语言，相应开发出三个需求分析支撑系统。

本书取材主要基于上述工作，但又不囿于此。撰写中力求系统化，并将作者近年学习心得贯穿于其中。俾读者能窥其源，见其流，冀收触类旁通之效。

1.5.2 组织按语言级别分篇

全书共十八章。除第一章引言外，其余十七章按语言级别归为四篇，第二章至第七章归为需求级语言篇，在概述软件需求分析之后，着重讲述作者自行设计之三种需求定义语言（NDRDL, NDRDL2.0, NDORL）与三个需求分析支撑系统（NDRASS, NDRASS 2.0, NDORASS）。第八和第九两章归为功能级语言篇，着重讲述功能规约语言 FGSPEC 之设计与实现。第十章至第十二章归为设计级语言篇，着重讲述设计规约语言 GSPEC 之设计、验证与实现。第十三章至第十八章归为实现级语言篇，分别讨论 ALGOL 60 与 ADA、系统程序设计语言、对象式程序设计语言及其形式语义、函数式语言以及逻辑式与函数式结合之语言等。

1.5.3 内容侧重语言，兼及实现

语言与实现之关系是，语言是实现之基础、实现之依据、实现之前提，语言理解不透，实现则难以启手。反之，实现作用亦不能低估，语言未经实现，则无法臻于实用，无法起其应有之作用，故而本书内容侧重语言，兼及实现。

“侧重”意指，对语言源流、设计背景、设计思想、设计原则、成分取舍、应用实况、利弊得失等均宜详细讨论，有观点、有原则、亦有例证，使之有血有肉，血肉相联，浑然成一整体。但力戒材料堆积、力戒“多多益善”，而应取材至精，存精华，而弃糟粕，区分本质与非本质。“兼及”则表明，实现讨论不应阙如，书中所论各种自行设计之语言均论及相应实现系统与应用实例，但其取舍亦应遵循“少而精”原则。“实践是检验真理之唯一标准”。语言未经实践，则难以判其优劣，难以为后继语言之设计提供有力佐证，非仅无法使用，亦难以对软件科学技术、计算机科学技术之发展有所促进。

本书撰写着重阐明思想，讲清原则与架构，不多涉及语言成分细节，以便于读者谙其本质、理顺关系、辨其优劣，一旦接触新语言或自行设计新语言时，能高屋建瓴、得心应手。目标如此，能否如愿，尚难逆料，吾等当竭尽全力以赴之。

需求级语言篇

第二章　软件需求分析概述

本章为软件需求分析概述，内容包括软件需求分析含义、软件需求定义、软件需求定义语言三节。

2.1　软件需求分析含义

软件需求分析是软件开发之第一阶段，是软件开发中最重要、最困难之阶段，其含义有二：其一是指从客户或用户提出之软件需求陈述出发，经过客户或用户与软件开发人员合作，将一般用自然语言表述之不完备、有歧义之需求完备化、一致化，形成可能采用半形式化、形式化语言表述之软件需求定义（亦称软件需求规约），使之作为客户或用户与软件开发人员间之契约基础，同时也作为后继开发阶段之依据。亦即，软件需求分析是收集澄清客户或用户需求，得出需求定义之过程。其二，它不仅指从客户或用户所提需求出发，得出软件需求定义，而且还包括从软件需求定义转换到相应软件功能规约之过程。细言之，软件需求分析又可区分为如下步骤，即，信息收集、规约形成（需求规约或需求规约与功能规约）、质量验证以及规约确认等步骤。信息收集是指，收集并澄清恰为客户或用户所需系统之必要需求信息；规约形成是指，将所收集之需求信息陈述成完整文档，一般称作软件需求定义或软件需求规约，或再将后者转换成相应软件功能规约；质量验证是指，保证软件需求定义或软件需求定义与软件功能规约之一致性、完备性；规约确认保证需求定义或软件需求定义与软件功能规约能精确描述客户或用户所需之软件需求。

实施软件需求分析之工程称为软件需求工程。

2.2　软件需求定义（规约）

2.2.1　含义

软件需求定义是软件需求精确而完整的陈述，它是以清晰、简明、一致且无歧义之方式刻画客户或用户所需系统所有重要方面的一组陈述，它含且仅含使设计人员与实现人员能生产出满足客户或用户所需系统之信息，它是软件开发人员与客户或用户密切合作、了解客户或用户需求、目的和期望，并进一步表述而成之定义性陈述，有时又称软件需求规约，它是客户或用户和分析人员之接口界面。

2.2.2　内容

软件需求包括功能需求与非功能需求两方面，其基本内容如下：

2.2.2.1 功能需求

它是描述系统功能之一组陈述。这些需求不仅包括所有外部可见之行为，还可能包括用以明显支持外部行为之所谓内在功能，不仅要阐明每项功能需要“做什么”，而且还须指明这些功能间之联系，以及相互间之依赖关系，但不涉及“如何做”之描述，功能需求是整个软件需求之核心。

2.2.2.2 非功能需求

它是在功能需求基础上，对软件需求进一步刻画，如功能限制、设计限制、数据与通信规程、系统目的、环境描述以及项目管理等。

第一，功能限制，在有些应用中，计算结果需满足一定限制，需求定义中即应包含这些限制，如性能、响应时间、用户数目、安全性要求、质量指标以及采用标准等。这些需求并不更动系统总体功能，但对实现却有影响。

第二，设计限制，除了功能限制外，客户或用户还会提出一些设计限制，其内容相当广泛，主要包括系统开发平台。如系统兼容性、硬件选择、操作系统选择等等，这些需求仅当客户或用户认为至关重要时，才纳入需求定义。

第三，数据与通信规程，多数系统都以某种方式和“外界”通信，这些通信规程需载入需求定义。

第四，系统目的与环境描述，如果系统设计人员与实现人员能了解系统开发背景与目的，则会使系统开发更为顺利。因而，系统目的与环境描述往往也含于需求定义中。

第五，项目管理，为了保证项目进展顺利，以及系统交付能使客户满意，需求定义中还宜包括诸如限期、预期更动、生存周期、安装细节、手册标准以及培训信息等。

2.2.3 目的

实现需求分析，客户或用户及分析人员便可更好了解待开发之系统，分析过程可促进客户或用户与开发人员相互了解、相互交流，并且可在技术上建立起良好合作关系。

但是，需求分析最重要之目的却是对软件开发后继阶段（如设计与实现）提供充分与必要之信息，以确保这些后继阶段能顺利完成。随着计算机系统愈加复杂，在其设计与构作中错误会明显增加。这一问题在不能容忍系统失效之系统（如安全性第一之系统）中显得尤为突出。如果能将错误避免或将之降到可被接受之最低限度，重要的则是，能开发出一组完备目一致之需求。否则，便可能忽略一些重要功能或功能限制。这就要求，一旦产生出需求文档，则总能保持其完备和一致，以使设计人员与实现人员能有牢固的工作基础。

开发人员需能确保所构作之系统与构作过程均正确，规约确认可增强开发人员信心，确信所产生之系统正确。需求定义正是验证所构作之系统是否正确构作之基准。对于有效确认和验证来说，清晰一致且完备之需求定义至关重要。

为确保项目进展顺利，项目管理要有完整之项目计划，其中包括价格、限期、资源、交付条件等等。由于一个完备之需求定义将包含上述信息，因而需求分析需详尽而精确，以使项目便于管理。同时，需求定义亦为接受性测试之标准。

2.2.4 使用

阅读与理解需求定义者有各类人员，如客户和用户、分析人员、设计人员、实现人员以及项目管理人员等，其各自背景与技术能力均有差异。客户和用户对需求之兴趣在于有效性，分析人员收集信息和形成需求定义，客户和用户作为信息源参与需求分析工作，一旦需求定义形成，客户和用户将对之进行审查，以确保需求定义恰能刻画他们所需之系统，在审查中客户和用户确认了需求定义，为使确认能顺利进行，需求定义之清晰性与便于理解极为重要。

在需求定义形成中，分析人员考虑其一致性、完备性以及正确性。一旦需求定义定型，设计人员与实现人员便要据此构作系统，分析人员之作用是确保系统得以正确开发。当然，这项任务之责任并不完全取决于分析人员。因此，分析人员对需求之兴趣是分析和解释。分析确保一致性与正确性，解释有助于完备性与系统开发。对分析人员而言，形式性（或精确性）颇为重要。分析仅在使用形式化语言描述时才能顺利进行，其无歧义性有利于保证分析人员之解释一致。

设计人员与实现人员根据需求定义设计与构作系统，他们对需求定义之兴趣只是将之视为蓝图，需求定义是系统开发之基础与准绳。设计人员与分析人员应尽量使用形式化语言，乃因为这类语言无歧义性，并具有一致性与简明性。从而，相对需求定义而言，系统验证工作将更简单可靠。

项目管理人员指导系统开发，以保证系统能及时交付、能符合可用标准、有完善文档等等，他们对需求定义之兴趣主要是项目管理信息。但是，他们也对保证系统交付日期感兴趣。因之，项目管理人员需在一定程度上熟悉全部需求定义，以便能确信，所交付之系统的确满足需求，的确满足客户所需。

2.2.5 现状

尽管近年来需求分析领域发展较快，但现状仍难令人满意，和上述讨论仍有相当差距，借助清晰性、一致性和完备性来反映需求特性难以奏效，需求定义用自然语言书写者居多，且篇幅巨大、自然语言之歧义性、验证与确认之困难、确保一致性与完备性中所遇问题，以及系统开发中各类人员间可能产生之误解等等，所有这些问题表明，在需求分析中使用自然语言并不合适，往往引起在需求定义中对系统的有些方面描述过多，而对另一些方面却又失之过少。阅读巨大之需求文档往往令人望而生畏，要完全理解用自然语言书写之需求定义十分困难，现状往往是，基于用正文表述需求定义之大型系统开发成为旷日持久、充满错误之实践。

然而，分析人员、设计人员以及实现人员也不责怪现实，他们仍是经常使用自然语言，这是因为舍此难有其它合适选择之故。虽然已有不少形式化语言，但却很少用于大型项目开发实践，设计人员与实现人员还是不太愿意使用，现有各种形式化语言很难表述全部功能需求与非功能需求，有些需求目前尚难用形式化语言表述，安全性需求即为一例。当然，就需求分析而言，亦有少量形式化语言可为客户和用户理解。需求定义所需之精确性要求使用形式化语言，而清晰性与易理解性却又要求抵制使用这类语言。

2.3 软件需求定义语言

2.3.1 定义

软件需求定义语言是用于书写软件需求定义之语言，简称需求定义语言。需求级语言即指需求定义语言。

按照形式化程度，需求定义语言可分为非形式化需求定义语言、半形式化需求定义语言以及形式化需求定义语言三类。

非形式化需求定义语言是指未加任何限制之自然语言，这类语言易学易用，易为用户接受，但由于自然语言之非形式性致使需求定义经常有错，且难以提供自动化支持；半形式化需求定义语言是指在宏观上对语言之语法和语义有精确描述，而在有些局部方面则允许使用非形式化自然语言，这类语言既便于用户表达和理解需求定义，又可在某种程度上便于用机器对需求定义进行管理和正确性检查；形式化需求定义语言是指其语法和语义均有精确定义之语言，这类语言一般具有良好的数学基础，易于用来分析需求定义性质，但却要求用户之数学素养较高，并且用以书写之需求定义较难理解。

2.3.2 研究内容

需求定义语言之主要研究内容有基本模型、语言结构、语法、语义、语用分析和转换系统等。

需求定义语言之基本模型是相应软件开发风范在语言中之具体体现，它是表达软件需求之基础，同时也决定了参与需求分析之各类人员的思维方式。功能分解模型和对象式模型是两类重要模型，二者各有短长，在功能分解模型下，语言用户通常将待解问题抽象成需要满足之功能，通过功能分解方法来表述系统各部分间之关系，以此刻画用户之系统需求；而在对象式模型下，语言使用者可较为直接地刻画现实世界模型，并在此基础上描述所需软件需求，致使相应需求定义易于理解、易于修改和易于复用，应用领域不同，合适之基本模型可能随之而异，而基本模型之不同对语言结构与相应支撑方法亦将产生影响。

语言结构提供刻画系统需求之具体手段，它主要包括基本模型描述、数据描述、控制描述、抽象机制以及项目相关信息描述等。当然，语言不同，侧重点会随之而异，在功能分解模型描述方面，SA(Structured Analysis)采用自顶向下、逐层分解之功能模型，为了支持该模型之描述，SA 提供了分层数据流图；而 PSL(Problem Statement Language)则采用自底向上途径，提供了各种实体类型及其相互“联系”，用以分别描述各个需求，然后形成所需之需求定义。当然，并非所有语言均基于单一模型之单一方法。例如， RSL(Real-time System Language)试图将自顶向下和自底向上途径结合，为了支持实时系统描述之需，提供了 R-网等设施。在对象式模型支持方面，通常有两种途径。其一是设计新型需求模型和语言来支持对象式方法。例如，D.W.Embley 等人提出之对象式需求模型，提供了各种图形化表示机制来刻画对象及其相互关系；其二是用现有语言来刻画对象式模型。例如，ROOA 方法采用形式规约语言 LOTO 中之抽象数据类型和进程定义来描述需求定义，并对继承机制提供支持；在数据描述方面， PSL 提供了 ENTITY, CONSISTS OF, DE-

RIVED BY 等实体或联系来描述数据对象名、数据结构和数据流程。在控制描述方面，PSL 提供了各种类似程序设计语言中之控制结构。尽管此类结构对于通信系统和实时系统较为有用，但却可能引起与实现有关之问题，在抽象机制方面， RML (Requirements Modelling Language) 语言在对象式架构下提供了聚合、分类和泛化三类抽象机制，它们可一致地用于语言中三类规约单位：对象、活动、断言。在项目相关信息描述方面，各类半形式化语言均提供了描述这些信息之手段。例如， PSL 可描述文档信息，这些信息通常是用自然语言来刻画的。

语法和语义之严格定义对需求定义语言之设计颇为重要，究竟采用何种形式体系来刻画语言之语法，采用何种途径来描述语言之形式语义，这些都是需求定义语言之重要研究课题。

语用分析主要讨论需求定义语言之适用领域、语言成分与使用者之关系、可扩展性等性质以及相应方法与工具之支持等。例如， PSL 主要适合商业应用，RSL 主要瞄准实时系统；而有些语言如 SL则是通用语言；可扩展性是指可在原有语言结构基础上定义新型语言构造之能力。在此意义下， PSL 和 RSL 本质上均为可扩展语言，为了充分发挥需求定义语言之作用，必须研究相应方法与工具，方法应反映获取需求之原则和步骤，以及如何根据需求开发相应程序，而工具支撑则对需求分析提供自动支持。在某种意义下，几乎所有语言均对如何形成需求提出建议和提示，但实用方法却不多见，就工具支撑而言， SA 是为手工使用而设计的，而 PSL, RSL 一开始就将工具支撑和语言设计联系起来。

如前所述，虽然需求定义语言主要面向用户，但是，用以书写之需求定义乃是软件后继开发之依据。因此，如能研制出旨在从需求定义之功能需求到相应形式功能规约之转换系统，则不仅可以提高软件生产率，而且亦可提高软件质量。因此，转换系统研究也是需求定义语言研究之重要方面。

需求定义语言之研究虽有较大进展，但仍有不少问题值得深入研究。第一，面向问题，按照目前理解，需求定义语言主要是面向用户，但用以书写之需求定义乃是后继开发阶段之依据。功能规约正确与否乃相对需求定义而言，如果功能规约能满足需求定义中之功能需求，便认为正确；反之，则认为不正确，问题在于，需求定义之作用只是作为软件开发后继阶段之依据文档，还是既作为文档，又作为由需求定义之功能需求转换到相应功能规约转换系统之输入。实际上，目前多数需求定义语言尚未配以相应可生产性使用之转换系统。从而，用以书写之需求定义主要还是只起文档作用，看来，这还相当不够。需求定义语言应有两个面向。即，既面向用户，又面向系统。第二，完备性问题，用户给出之原始需求陈述很难完备，这里既有客观原因也有主观原因，客观原因是，对问题有一认识过程，亦即，人们认识问题有一由不完备到完备过渡之过程；主观原因则是，用户给出之需求陈述是否完备往往和用户知识背景与水平有关。事实上，这一完备化过程乃是通过用户与分析人员反复交互、多次迭代完成的，而且多数情形是，完备化过程同时也是精确化过程与形式化过程。这就是说，由不完备到完备，由不精确到精确，由非形式化、半形式化到形式化，三者往往统一在一个过程之中。问题在于，完备与否，精确与否，总是相对的，需求定义完备是指它能全面反映用户需求；需求定义语言完备是指它包含且仅包含所有赖以书写各种完备需求定义之语言成分；需求定义精确是指它本身不存在歧义，不存在不一致；需求定义语言精确是指其语言成分不存在歧义，各成分间不存在不一致。这些问题

都需要在设计需求定义语言之需求定义结构及其各个成分之结构时仔细斟酌，多加思考。第三，正确性问题，用需求定义语言书写之需求定义是否正确，当然十分重要，需求定义一旦有误，后继开发必将不正确。但是，何谓需求定义正确，目前检验方法是，一看它是否符合相应语言语法，二看它本身是否完备与精确。当然，最好是通过相应检验程序自动进行。

第三章 软件需求定义语言 NDRDL

本章讨论软件需求定义语言 NDRDL,内容包括设计目标、设计原则、语言成分、讨论以及功能构造之形式语义五节。

南京大学计算机软件研究所研究软件自动化近二十年，先后开发出 NDAUTO, NDADAS, NDSAIL 等多个设计级与功能级软件自动化系统。考虑到需求分析阶段之重要性，近年来又致力于研究需求分析支撑系统，用以支持需求分析阶段工作，特别是，试图将用需求定义语言书写之需求定义中功能需求在一定意义下自动转换成相应软件功能规约。为此，就必须考虑需求定义语言。鉴于现有各种需求定义语言或者偏于专用，或者形式化程度较低，经过调查研究，设计并实现了需求定义语言 NDRDL，以此作为软件需求分析支撑系统 NDRASS 之源语言。

3.1 设 计 目 标

3.1.1 实用性佳

需求定义语言是用以书写需求定义之语言。良好之需求定义应尽可能详细、完备、一致，并且适用性较强，它在软件开发中举足轻重。对软件开发人员而言，用以书写之需求定义不仅是后继开发阶段之依据，而且也是软件验证和确认之基础；对用户而言，它则是用户和软件开发人员间建立契约之基础。因而，在软件需求定义语言设计中必须讲究实用，惟有如此，用户和开发人员才会乐于使用。所以，实用性佳为 NDRDL之第一设计目标。

3.1.2 表述力强

其次，由于需求定义是软件需求之完整陈述。它既包含功能需求又包含非功能需求。因此，需求定义语言既应具有表述功能需求之成分，又应具有表述非功能需求之成分。亦即，需求定义语言中应包含表述完整需求定义之成分。所以，表述力强是 NDRDL 之第二设计目标。

3.1.3 易读性好

第三，需求定义语言不同于其它级语言之处还在于，它主要面向用户。所以，其易读性应较好。否则，语言所描述之需求定义将使用户不易阅读，从而也就较难推广使用。因此,易读性好是 NDRDL 之第三设计目标。

3.1.4 严谨性高

第四，需求定义作为后继开发阶段之依据，需求定义语言又应具有严格之语法和语义。否则，不仅难以描述完整需求定义，而且会导致软件需求分析阶段结果错误较多，从