Programming Languages | B. Wegbreit Editor

Programming Languages, Natural Languages, and Mathematics

Peter Naur
Datalogisk Institut, Copenhagen University

Some social aspects of pro gramming are illuminated through analogies with similar aspects of mathematics and natural languages. The split between pure and applied mathematics is found similarly in programming. The development of natural languages toward flexionless, word-order based language types speaks for programming language design based on general, abstract constructs. By analogy with incidents of the history of artificial, auxiliary languages it is suggested that Fortran and Cobol will remain dominant for a long time to come. The most promising avenues for further work of wide influence are seen to be high quality program literature (i.e. programs) of general utility and studies of questions related to program style.

Key Words and Phrases: analogies related to social aspects, pure and applied mathematics, language quality, language development, artificial auxiliary languages, literature, style, descriptive and prescriptive attitudes

CR Categories: 4.20

1. Introduction

Programming has many points in common with both mathematics and natural languages. In these notes I shall not try to cover all of these, but I shall be particularly concerned with these fields as they exist and are used by groups of people. Perhaps a reasonably accurate description of my main topic is: What can we learn about programming languages from the social aspects of mathematics and natural languages?

Now why is this a relevant topic to consider? Well, I definitely feel that there is something rotten in the realm of programming. There is a lot of discussion, but somehow I think that most of it misses the point. There are too many fads, too many quick solutions, a too wide gap between theory and practice. I feel that we badly need to take stock. But before doing so we need to widen our horizon. This is where mathematics and natural languages come in. They are our closest relatives among the intellectual disciplines, but as a major point of difference, they are old. They have lived in this world for so long that they provide a long series of cases of confrontation with people, with society, cases from which we might learn something. This obviously is a vast undertaking, if it is to be done thoroughly, far beyond what I am capable of carrying through. What I can do at present is no more than discuss a series of points that I have noted over the last many years and that I have found illuminating. I pretend to no more than that. In particular, I use a number of sources as I have found them. I will give you the references to them, but whether they have been superseded or found wrong by later researchers I do not always know for sure. So I hope to stir your thoughts, but I will not pretend that I have found the answers.

2. Pure and Applied Mathematics

Now, to start with mathematics, I think it is proper to say at once that I have never been a mathematician, or even hoped to become one. Early in my life I was caught by astronomy, which became my chosen field for many years. I quickly realized that to go ahead in this field I had to know some mathematics, which I then proceeded to study. But mathematics remained for me at all times a tool, not something of primary, intrinsic interest. With this point of departure it will not be too surprising that I look upon much of the mathematical activity with considerable scepticism as to its importance. I certainly cannot view mathematics as an ideal that programming should strive against. On the other hand, I do see a great similarity between mathematics and programming, first and foremost because they both are tools for a wide variety of other disciplines. So this is the point I want to consider, the relation of mathematics to the disciplines that use it as a tool.

For many years there has been an acknowledged distinction between pure and applied mathematics. This distinction seems to settle the relation to applications in a clear and simple manner. The point I want to make is that this is by no means as simple as it looks from outside. If we try to pursue this matter we may seek the opinion of the mathematicians themselves.

A version of this paper was presented at the Second ACM Symposium on Principles of Programming Languages, Palo Alto, Calif., Jan. 20–22, 1975.

Author's address: Datalogisk Institut, Copenhagen University, Sigurdsgade 41, DK-2220 Copenhagen N, Denmark.

This may reveal some real divergencies of view. At one end of the spectrum we may find G. H. Hardy (1877–1947), one of the leading British mathematicians of his day. In his book *A Mathematician's Apology* [1940] he presents the view of a pure mathematician who will defend his chosen field to the point of provocation:

"I have never done anything 'useful'. No discovery of mine has made, or is likely to make, directly or indirectly, for good or ill, the least difference to the amenity of the world." "A mathematician, like a painter or a poet, is a maker of patterns. . . . The mathematician's patterns, like the painter's or the poet's, must be *beautiful*; the ideas, like the colours or the words, must fit together in a harmonious way. Beauty is the first test: there is no place in the world for ugly mathematics." ". . . very little of mathematics is useful practically, and that little is comparatively dull."

How widespread Hardy's view is among mathematicians I don't know for sure. There are, however, other views. John Von Neumann, in an article titled "The Mathematician" [1947], expresses a rather different outlook:

"The most vitally characteristic fact about mathematics is, in my opinion, its quite peculiar relationship to the natural sciences, or, more generally, to any science which interprets experience on a higher than purely descriptive level.

"Most people, mathematicians and others, will agree that mathematics is not an empirical science, or at least that it is practised in a manner which differs in several decisive respects from the techniques of the empirical sciences. And, yet, its development is very closely linked with the natural sciences. One of its main branches, geometry, actually started as a natural, empirical science. Some of the best inspirations of modern mathematics (I believe, the best ones) clearly originated in the natural sciences." . . . "There is a quite peculiar duplicity in the nature of mathematics. One has to realize this duplicity, to accept it, and to assimilate it into one's thinking on the subject. This double face is the face of mathematics, . . . "

So much for John von Neumann.

The question is more subtle than these two quotations indicate, however. Marshall Stone [1961], in an article titled "The Revolution in Mathematics," stated, on the one hand:

"While several important changes have taken place since 1900 in our conception of mathematics or in our points of view concerning it, the one which truly involves a revolution in ideas is the discovery that mathematics is entirely independent of the physical world. . . . When we stop to compare the mathematics of today with mathematics as it was at the close of the nineteenth century we may well be amazed to note how rapidly our mathematical knowledge has grown in quantity and in complexity, but we should also not fail to observe how closely this development has been involved with an emphasis upon abstraction and an increasing concern with the perception and analysis of broad mathematical patterns. Indeed, upon closer examination we see that this new orientation, made possible only by the divorce of mathematics from its applications, has been the true source of its tremendous vitality and growth during the present century."

On the other hand, later in the same article, when discussing applications of mathematics, we find:

"For it is only to the extent that mathematics is freed from the bonds which have attached it in the past to particular aspects of reality that it can become the extremely flexible and powerful instrument we need to break paths into areas now beyond our ken."

Thus we have here a claim that precisely in being pure, mathematics will best serve the applications!

As an example of the kind of insight that is gained through considerations at a high level of abstract generality, here is Stone's characterization of our field:

"Indeed, one can see—after the event—that the entire development of modern machine methods in mathematical computation has been made possible by two basic advances—Russell and Whitehead's discovery of the reduction of mathematics to formal logic and the invention of the vacuum tube."

Stone's statements soon met stern reprobation. In a panel discussion in November 1961 Richard Courant makes Stone's article his point of departure, saying:

". . . as a sweeping claim, as an attempt to lay down a line for research and before all for education, the article seems a danger signal, and certainly in need of supplementation. The danger of enthusiastic abstractionism is compounded by the fact that this fashion does not at all advocate nonsense, but merely promotes a half truth. One-sided half truths must not be allowed to sweep aside the vital aspects of the balanced whole truth. Certainly mathematical thought operates by abstraction; . . . Yet, the life blood of our science rises through its roots; these roots reach down in endless ramification deep into what might be called reality. . . . Abstraction and generalization is not more vital for mathematics than individuality of phenomena and, before all, not more than inductive intuition. . . . We must not accept the old blasphemous nonsense that the ultimate justification of mathematical science is 'the glory of the human mind'. Mathematics must not be allowed to split and to diverge toward a 'pure' and an 'applied' variety. It must remain, and be strengthened as, a unified vital strand in the broad stream of science and must be prevented from becoming a little side brook that might disappear in the sand. The divergent tendencies are immanent in mathematics and yet prove an ever-present danger.

To this it may be added that historically the split between mathematicians pursuing the pure and applied branches is fairly new. Up to and including the time of Gauss, who lived from 1777 to 1855, the mathematicians were active in both directions. The split may have something to do with the specialization that became a necessity when the field became too vast for any individual to master it all. Another likely contributing circumstance is the modern type of university. Indeed, pure mathematics, with its limitless scope for the creation of conceptual structures of incontestable coherence, but practical irrelevance, is a perfect field of study for a self-governing academic institution. However, again this split becomes a problem only when the idea starts to spread that one kind of mathematics is inferior to the other. Sad to say, that is where we are today.

Now, do these problems and dangers carry over into the field of programming? As far as I can see, to a considerable extent. Von Neumann's "double face of mathematics" has a direct analogy in programming, where we have the duplicity of, on the one hand, programming as "an intellectual challenge" (E.W. Dijkstra, 1972), or as a field for never-ending refinement of programming language facilities, and on the other hand, programming applied to a wide range of practical problems by thousands of programmers. And the split described by Courant is clearly there with us. The second NATO Software Engineering Conference [Buxton and Randell 1969], found this to be the core question.

Accepting then that the analogy between mathematics and programming is valid on this issue, what may we learn from mathematics? The obvious lesson is that the deep gap between theory and practice in programming may persist for long periods without anything

Otto Jespersen. From a painting by Julius Paulsen.

much being done about it. The danger envisioned by Courant, of mathematics "becoming a little side brook that might disappear in the sand" has not so far turned out to be real. It was brought up by Courant as a threat that might make people work more seriously on removing the split of mathematics. But, as far as I know, it has had little or no effect. I see no reason why programming should fare differently. Already today we have an established pattern: on the one side, the universities immersed in computer science projects of a highly esoteric nature, in only very slight contact with the activity of the other side, the computer industry, producing vast amounts of programming that to the university people are very unattractive. I see no reason why this pattern will not persist, whether we like it or not.

3. Programming Languages and Natural Languages

Let me now turn to my second field of inspiration, natural language. At this point of transition I cannot help expressing a feeling of awkwardness at the use of the word language in the context "programming language." I definitely feel that if taken literally this habit of expression is misleading. As a first step to subdue our feeling of guilt at this misuse of the term, perhaps we can remind ourselves that logicians and mathematicians used the word for specialized notations long before we did, already in the 1930's, and perhaps even before.

However, I feel urged to clarify the relation further. Thus, although I find several analogies that are illuminating, I do not wish to overlook the differences of characteristics and use between natural and programming languages. Among such differences I would like to mention:

(1) Natural languages are used mostly in spoken form, in conversations. Humans possess extraordinary capabilities in handling spoken language. They perceive rapidly what they need from a highly complex stream of verbal sounds. They are accustomed to making use of the interactive possibilities of conversations, in clearing up what they have not understood. By contrast, programming is done mostly in writing and in terms of a slow and clumsy interaction with a computer. This difference in principal mode of use certainly must be taken into account if we try to transfer experience from the field of natural language into programming.

(2) Apart from certain exceptional uses of natural languages, for example in science, most of what is said in such languages is intended and understood only in a fuzzy way, conveying roughly the speaker's subjective reaction to some not very precisely defined circumstance of the world. By contrast, anything expressed in a programming language is intended to be understood and interpreted by a computer in a perfectly well-defined way. This difference between natural language and programming cannot lightly be brushed aside by claiming that the lack of precision of natural language is a defect that can and should be removed. On the contrary, it seems quite possible that this lack of precision is vital to the ability of natural language to develop and make it possible to speak of a never ending series of ever new ideas. From this point of view every programming language is just another example of the ability of natural language to extend itself to cover new ground. Thus there is no deep analogy between a natural language and a programming language. Rather, there is a relation of subordination. Everything that can be expressed in a programming language can be expressed equivalently, although perhaps clumsily, in a host natural language, while the converse is utterly false. A natural language provides an extensible universe of discourse that allows new notations, such as programming languages, to be added at will.

With this warning against rash conclusions behind me I shall now get down to the analogies that I still find illuminating.

Drawing analogies between natural languages and programming languages is already an old sport. Just think of Chomsky's three types of language. However, the analogies that I shall pursue have not received much attention before, as far as I know. Many of them will be derived from the work of Otto Jespersen (1860–1943), the great Danish scholar whose works on the English language and its grammar continue to be used wherever English is studied seriously. For many years I have found Jespersen's writings a rich source of enjoyable enlightenment. He took a very broad view of his chosen field and combined this with a bent for polemics. During his early days he became a leader of a

reaction against an earlier school of linguists who fed on an endless delight in the classical languages, Latin above all. From his earliest work, around 1890, Jespersen maintained the superiority of modern languages, with English and Chinese at the top of his scale. At this time he was also very active in furthering the young field of phonetics. As one outcome of his interest he contributed very actively to the writing of elementary textbooks of English for use in Danish schools. In his later years his major contribution was a monumental Modern English Grammar in seven volumes. This did not prevent him from also taking active part in the work on establishing an artificial international language. Several of these fields of activity offer interesting analogies in the present context.

4. Language Quality

The question of the quality of a natural language is treated at length by Jespersen. More specifically he is concerned with linguistic development, and whether the development that natural languages have undergone should be regarded as progress or decay [Jespersen 1922, pp. 319–366]. Jespersen first wrote about this question in 1894 at a time when linguistics was dominated by an interest in and veneration for classical Latin and Greek. The attitude of this period is later described thus by Jespersen [1922]:

"To men fresh from the ordinary grammar-school training, no language would seem really respectable that had not four or five distinct cases and three genders, or had less than five tenses and as many moods in its verbs. Accordingly, such poor languages as had either lost much of their original richness in grammatical forms (e.g. French, English, or Danish), or had never had any, so far as one knew (e.g. Chinese), were naturally looked upon with something of the pity bestowed on relatives in reduced circumstances, or the contempt felt for foreign paupers."

Jespersen criticized this attitude, first, as being vague and lacking in support from instances collected systematically. Second, as being unclear as to

"the method by which linguistic value is to be measured, by what standard and what tests the comparative merits of languages or of forms are to be ascertained."

He then goes on to describe his own yardstick:

"What is to be taken into account is of course the interests of the speaking community, and if we consistently consider language as a set of human actions with a definite end in view, namely, the communications of thoughts and feelings, then it becomes easy to find tests by which to measure linguistic values, for from that point of view it is evident that THAT LANGUAGE RANKS HIGHEST WHICH GOES FARTHEST IN THE ART OF ACCOMPLISHING MUCH WITH LITTLE MEANS, OR, IN OTHER WORDS, WHICH IS ABLE TO EXPRESS THE GREATEST AMOUNT OF MEANING WITH THE SIMPLEST MECHANISM."

At this point Jespersen discusses an objection to this attitude, based on a claim that natives handle their language without the least exertion or effort. On the basis of several kinds of observations he concludes:

"To my mind there is not the slightest doubt that different languages differ very much in easiness even to native speakers."

Jespersen now sets out to consider the development of natural languages, and in particular, to find out whether this development may be regarded as one of progress or decay if measured along his chosen scale of linguistic values. His conclusion is as follows:

"The sum total of linguistic changes, when we compare a remote period with the present time, shows a surplus of progressive over retrogressive or indifferent changes."

This conclusion follows upon a lengthy discussion that Jespersen summarizes in seven points. These points, where modern languages manifest their superiority over earlier language forms, with a few examples added for illustration, are:

(1) The forms are generally shorter, thus involving less muscular exertion and requiring less time for their enunciation. Examples: Old English *lufu*—English *love*; Latin *augustum*—French *août*; Gothic *habaidedeima*—English *had*. A global indication: the Gospel of St. Matthew contains in Greek about 39,000 syllables, in German 33,000, in English 29,000.

(2) There are not so many forms to burden the memory. Example: Gothic *habaida, habaides, habaidedu,* and twelve further forms are all rendered in English by *had.*

(3) The formation of word forms is much more regular.

(4) Their syntactic use presents fewer irregularities.

(5) Their more analytic and abstract character facilitates expression by rendering possible a great many combinations and constructions which were formerly impossible or unidiomatic. Example: in Danish there is no difficulty in saying: 'enten du cller jeg har uret'; this causes trouble in languages with inflected verbs, even in English: 'either you or I are wrong', 'either you or I am wrong' are both incorrect; 'either you are wrong, or I' is stiff and awkward.

(6) The clumsy repetitions known under the name of concord have become superfluous. Example: In Latin 'opera virorum omnium bonorum veterum' plural number is expressed four times, genitive case also four times, and masculine gender twice; in the corresponding English 'all good old men's works' each idea is expressed once only.

(7) A clear and unambiguous understanding is secured through a regular word order. Example: 'John beats Henry' and 'Henry beats John' have two different, clear meanings. In Danish two further arrangements of the same three words have new, clear meanings: 'Slår Henry John?', 'Slår John Henry?'.

The relevance of these results to programming appears to me to be the manner in which they reveal what is, in a deep sense, a natural form of expression for human beings. After all, these trends reflect tendencies of millions of individuals over centuries! They may well be taken as guidance in designing programming languages. They support the idea that programming languages should preferably be built from a few, very

general, very abstract concepts, that can be applied in many combinations, thereby yielding the desired flexibility of expression. They provide an argument against programming languages that include many special features with limited fields of application, and that include many exceptional cases in their definition.

Point (5), concerning freedom of combinations of the language elements, would seem to support a freedom from, for example, type restrictions. It might perhaps support the attitude taken in PL/I where, as far as possible, every combination of operands and operators has a meaning, in opposition to the attitude of, for example, Algol 60 and Simula, where type restrictions make many combinations illegal. However, in judging this point the difference between a language for easy interactive use and one predominantly for one-way communication should perhaps be considered. The point may be made that the risk of misunderstanding due to a mistake that is introduced when free combinations are allowed, can be tolerated only in a system that includes rapid and effective interactive check of the correctness of more unusual combinations.

The replacement, in natural languages, of concord by word order has an analogy in a replacement of program constructions based on labels and go-to-statements by constructions based on ordering of program structures. Thus the current interest in program structuring finds some support in the development of natural languages.

5. Artificial Auxiliary Languages

Let me now turn to another aspect of natural languages, and again one which has been taken very seriously by Otto Jespersen, namely artificial languages designed for international communication. In this field one finds striking analogies to the activity of programming language development. Perhaps the most outstanding point of analogy is one that lies behind a remark made by Jespersen at the point in his autobiography [1938, p. 126] where he starts telling about his activity in this area:

"This led to what many people will perhaps regard as the most stupid thing of my life (from time to time I am inclined to agree with them), namely the colossal amount of work I have spent on this cause . . .".

The point here is the way in which people get emotionally involved in the issues. Having myself taken part in some of the committee work on programming languages, I was struck by the similarity of the problems that arose around 1910 concerning the choice of an international language, as described by Jespersen.

This is not the place to cover the complete history of artificial languages. Such a history may be found, for example, in Bodmer [1944]. I will concentrate on two issues. The first is the conflict between language quality and language support. To bring this out, consider first the situation as it was around 1910. At this time the practicality of an artificial language was fully recognized. Since 1880, when Volapük started its hectic and brief career, a whole series of languages had been developed. One of them, Esperanto, already had ardent supporters, organized internationally. Another one, Interlingua or Latino Sine Flexione, developed by the mathematician Peano, was supported by the Academia pro Interlingua. At the turn of the century, a sufficient number of scholars had become interested in the whole question to make it possible to organize an international committee with the task of deciding on the best language. This committee met in Paris in 1907, including Jespersen as one of the two experts in linguistics. The committee decided that although Esperanto was a strong contender, there were so many details of it that were known to be in need of improvement that it could only be recommended on the condition that it was changed in a number of ways. The desirable directions of these changes were already brought forth in the language Ido that was first announced during the conference.

The deeply significant experience that may be derived from this incident is the reactions that ensued from the committee recommendation. To put it briefly, there came a fight. The Esperantist organizations refused to consider changes to their language. Meanwhile many individual adherents changed their support and worked on Ido and its further improvement. Emotions rose, declarations and insinuations flew in all directions. And thus it happened that even a language like Esperanto, that was recognized already in 1907 when it was but 20 years old to be poorly suited for its declared purpose, has retained its support to this day, 67 years later. This is a striking example of how people tend to shift their support from a higher level of generality, in this case an international language as such, to a lower, in this case the particular language that they have for some reason become attached to.

As a contrast to this misguided loyalty we may again turn to Otto Jespersen. He first contributed to the discussion of Ido, the modified form of Esperanto. Then came the First World War, which stopped the work. However, years later, in 1928, Jespersen published a wholly new attack on the problem, in the form of a description of a language called Novial [Jespersen 1928]. He followed this, two years later, with a complete dictionary of the language [Jespersen 1930]. And again, in 1934 when he was 73 years old, he published a critical reexamination of his own creation, proposing a number of modifications.

The analogy of these incidents with the development of programming languages in recent years seems to me close enough to be of interest. The basic lesson seems to be that where public appeal is concerned the technical quality of an artificial language is decisive only at a very early stage of development. Volapük did fall flat because of technical defects. As soon as the most glaring defects

have been removed, the public following of a language depends on the support around the language, not on the language itself. On this analogy, Esperanto corresponds to Fortran and Cobol. If this analogy is valid we should expect Fortran and Cobol to be with us for many years to come. As another conclusion, the discussions concerning the best features to include in a programming language will remain academic exercises, perhaps interesting, but without deep influence on the main course of events.

This sounds like a cry of despair. Is there nothing we can do to influence programming at large? Looking again to the hero of this story, Otto Jespersen, we may ask, what part of his activity has had the widest influence? This, of course, cannot be answered easily or precisely. Undoubtedly his monumental, seven-volume grammar of English and other great studies have had influence on linguistics. But, one may wonder, is this influence greater than that of his work on the elementary teaching of English in Danish schools? He was coauthor of a set of textbooks that were first published around 1895, just before the revolutionary shift from Latin to correctly pronounced English occurred in Danish schools. Countless Danish children have been introduced first to English through the display of powerful English monosyllables in the first sentence of these books: 'ai kən hʌp, ai kən rʌn, si' mi' hʌp, si' mi' rʌn, it iz fʌn, fʌn, fʌn!'. (For the clue to the phonetic script, see the end of the article[1]). Personally I am deeply indebted to the tradition starting from this work of Jespersen.

By analogy this stresses the importance of producing good textbooks for elementary teaching. However, this should not be overstressed since other circumstances than the existence of good textbooks undoubtedly have been at work in the revolution of English teaching in Denmark.

6. Fortran and Cobol as Given Facts of Life

Suppose that the analogy of Esperanto and Fortran plus Cobol is valid, in other words that Fortran and Cobol are here to stay. Then it would follow that the attempts to replace them by something better will fail.

This prospect was recently considered by Ralston [1973], who suggested that those who criticize Fortran would make a better contribution if they would work on improving this language and its use.

I feel that this point is too strong to be ignored. However, unlike Ralston I don't believe that those people who now oppose Fortran and work on other languages have much chance of influencing it. The trouble is that each of the languages, Fortran, Cobol, Algol 60, Algol 68, Simula 67, PL/I, APL, etc. have their separate maintenance committee. Such an arrangement is, by its very structure, bound to keep refueling adverse emotions. Each separate group by definition has as its task to guard that holy grail that is mentioned in the name of the group. In addition, the people of each group naturally will have the ambition to achieve something. This achievement consists of revisions of the language. But to enhance the sense of achievement it is important that revisions have a competitive value, that they contribute to the unique quality of the language. For this reason, when the members of the maintenance committee for language Q bring out a revision of their language they will never, never tell that this revision has been taken over from good features known for a long time in languages R and T. On the contrary, they will try to conceal this fact. Thus they will call them by new names and may well try to modify the features so as to make the influence less obvious. This process often will result in an "improvement" which impairs the features in question.

However, even if the influence on the languages as such are bound to rigid patterns, there is another point of influence, that of literature. This will be considered next.

7. Language and Literature

The point where the analogy of natural language and any programming language fails most miserably is that of literature. The field of programming just does not have anything that is comparable with the works of Shakespeare. While this fact is historically easy to understand, it suggests also that there might be a need for a shift of attention away from programming languages toward the corresponding literature, i.e. toward the programs written in the languages. With such a shift the focus of interest becomes algorithmic methods of general interest. Work in this direction must first of all aim at the writing of programs. In order to avoid a onesidedness that might turn the attention away from the proper focus, this type of work is at present best done if the algorithms are expressed in at least two different languages at the same time. Examples of work along this direction are the books by D. E. Knuth [1968–73] where the algorithms are expressed both in an informal programming language based on natural language and in the language MIX, and the recent book by the present writer [Naur 1974a] where Fortran and Algol 60 are used side by side.

When a suitable body of literature has become available, other forms of linguistic study will become pertinent. One such form is concerned with the style of programs. Work in this direction has already appeared [Weinberg 1970; Kernighan and Plauger 1974].

The interest in programming style is directly analogous to a corresponding interest in natural language style. Kernighan and Plauger [1974] directly point to a book on natural language style [Strunk and White, 1959] as a major source of inspiration. The most basic fact about natural language style in this context is that good

writing is very difficult, even for persons who have complete mastery of everyday spoken language. Good style is achieved only through insight, practice, and effort. By analogy we can expect that good programming style will remain a combination of sound principles, talent and work, and that the fight against poor style is never ending.

8. Descriptive and Prescriptive Attitude to Language

The dominant attitude of both laymen and scholars to language as such has until fairly recently been prescriptive, i.e. the interest has focused on what is regarded as correct, good language, and on the correct spelling. Enormous amounts of time and trouble have been spent on trying to stop people from using a preposition "to end the sentence with," from splitting their infinitives, etc.

In scholarly linguistics the prescriptive attitude has now largely been replaced by a descriptive attitude. Thus Jespersen based his great English grammar and other studies on observations of language as it is spoken and written. He estimates that throughout his life he has made 3–400,000 separate notes of his observations.

The prescriptive attitude is of course well justified in the case of beginners who are introduced to a new language, or are picking up the native language as young children. However, the prescriptive attitude makes itself strongly felt far outside the classroom and the nursery, and the motivation for this attitude goes far beyond the interest in unambiguous, effective conversation. One such extraneous prescriptive principle is that earlier forms or meanings of words should a priori be considered more correct than the current ones. Another extraneous motive of this kind is derived from the use of language for distinguishing classes of people. Bernard Shaw, in *Pygmalion*, gives a vivid picture of this verbal class distinction at work in England at the turn of the century.

There is little doubt that many of the prescriptive efforts that derive from an insistence on keeping older forms, or from the use of language for class distinction, have worked against the change of language in the direction of higher effectiveness. For example, in Danish written language a distinction between verbs used with singular and plural subjects was maintained until the end of the nineteenth century (e.g. jeg rejser = I travel, vi rejse = we travel). When this superfluous distinction was finally removed in the written language it had been dead in the spoken language for some three hundred years. One shudders to think of the countless hours that little girls and boys in Danish schools must have wasted in learning this rule. It served no useful purpose whatsoever. On the contrary, getting rid of the distinction is in some cases a direct help to unhindered expression.

In programming we find close analogies to these states of affairs. Thus the teaching of programming and programming languages is wholly prescriptive. We also find a prescriptive attitude that opposes innovations as such, and one that deprecates particular modes of expression, for example go-to statements.

On the point of language change, programming has a greater inertia than natural language because a change, to be really effective, has to be implemented in a compiler. For this reason the play with language, by way of expressions that perhaps start purely as jokes, is less active as a source of change. Instead programming language change goes by way of a series of new languages, deliberately designed as more or less convincing wholes. In presenting a new language to the world the designers must assume a predominantly descriptive attitude to their creation. A descriptive attitude has also been taken in a number of comparative studies of programming languages [e.g. Higman 1967; Caracciolo di Forino 1968].

In relation to program literature, the descriptive attitude has so far been almost completely absent, for the good reason that so little literature worthy of systematic study exists. The little that has been published in this direction has dealt with the behavior of programmers during the program development activity [e.g. Gold 1969; Naur 1974b]. This area of empirical studies of programming might well turn out to gain in significance in the future.

9. Conclusion

Several of the social aspects of mathematics and natural languages show a meaningful analogy with similar aspects of programming languages. It therefore makes sense to extrapolate the analogy to further such aspects. On this basis the following conclusions may be drawn: the split between the more academic, pure computer science oriented study of programming languages and the world of practical programming will persist indefinitely; the era of influential programming language construction is past, Fortran and Cobol will retain their dominance; the existing programming languages will develop slowly, with only weak interaction among them; the areas of widest influence from scholarly studies on programming at large will be program literature and style, the most important medium of influence being textbooks at the fairly elementary level; greater interest in empirical studies of programming may be expected to develop in the future.

Acknowledgements. The author wishes to thank the referee for his or her very helpful suggestions.

[1] Clue to the phonetic sentence in Section 5: 'I can hop, I can run, see me hop, see me run, it is fun, fun, fun!'.

References

Bodmer, F. [1944], with Hogben, L. (Ed.). *The Loom of Language.* George Allen and Unwin, London, 7th impr., 1961.

Buxton, J.N. and Randell, B. [1969]. Software Engineering Techniques. NATO Science Committee, Brussels, 1970.

Caracciolo di Forino, A. [1968]. String processing languages and generalized Markov algorithms. In *Symbol Manipulation Languages and Techniques,* D. G. Bobrow (Ed.), North Holland Pub. Co., Amsterdam.

Courant, R. [1961]. In Applied mathematics: what is needed in research and education—a symposium. *SIAM Rev. 4,* 4 (Oct. 1962), 297–320.

Dijkstra, E.W. [1972]. The humble programmer. *Comm. ACM 15,* 10 (Oct. 1972), 859–866.

Gold, M.M. [1969]. Time-sharing and batch processing: an experimental comparison of their values in a problem-solving situation. *Comm. ACM 12,* 5 (May 1969), 249–259.

Hardy, G.H. A Mathematician's Apology. Cambridge U. Press, reprinted 1967. Partially reprinted in *The World of Mathematics,* J.R. Newman (Ed.), Simon and Schuster, New York, 1956, pp. 2027–2038.

Higman, B. [1967]. *A Comparative Study of Programming Languages.* American Elsevier, New York.

Jespersen, O. [1922]. *Language, Its Nature, Development, and Origin.* George Allen and Unwin, London, 10th impr., 1954.

Jespersen, O. [1928]. *An International Language.* George Allen and Unwin, London.

Jespersen, O. [1930]. *Novial Lexike.* George Allen and Unwin, London.

Jespersen, O. [1938]. *En Sprogmands Levned.* Glyndendal, Copenhagen.

Kernighan, B.W. and Plauger, P.J. [1974]. *The Elements of Programming Style.* McGraw-Hill, New York.

Knuth, D.E. [1968–1973]. *The Art of Computer Programming.* Addison-Wesley, Reading, Mass.

Naur, P. [1974a]. *Concise Survey of Computer Methods.* Studentlitteratur, Lund, Sweden, and Petrocelli Books, New York.

Naur, P. [1974b]. What happens during program development—an experimental study. In *Systemering 75,* M. Lundberg and J. Bubenko (Eds.), Studentlitteratur, Lund, Sweden, pp. 269–289.

Ralston, A. [1973]. The future of higher-level languages (in teaching). In *International Computer Symposium 1973,* A. Gunther, B. Levrat, and H. Lipps (Eds.), American Elsevier, New York, 1974, pp. 1–10.

Stone, M. [1961]. The revolution in mathematics. *Amer. Math. Monthly* (Oct. 1961), 715–734.

Strunk, W. Jr., and White, E.B. [1959]. *The Elements of Style.* Macmillan, New York.

Von Neumann, J. [1947]. The mathematician. In *The Works of the Mind,* Heywood and Neff (Eds.), U. of Chicago Press. Also in *The Collected Works of John Von Neumann,* Vol. 1, Princeton U. Press, pp. 1–9; and in *The World of Mathematics,* J.R. Neumann, (Ed.), Simon and Schuster, New York, 1956, pp. 2053–2063.

Weinberg, G.M. [1970]. *PL/I Programming: A Manual of Style.* McGraw-Hill, New York.

Programming Languages

B. Wegbreit Editor

Exception Handling: Issues and a Proposed Notation

John B. Goodenough
SofTech, Inc.

This paper defines exception conditions, discusses the requirements exception handling language features must satisfy, and proposes some new language features for dealing with exceptions in an orderly and reliable way. The proposed language features serve to highlight exception handling issues by showing how deficiencies in current approaches can be remedied.

Key Words and Phrases: multilevel exit, goto statement, error conditions, structured programming, ON conditions, programming languages

CR Categories: 4.22

1. The Nature of Exception Conditions

Of the conditions detected while attempting to perform some operation, *exception conditions* are those brought to the attention of the operation's invoker. The invoker is then permitted (or required) to respond to the condition. Bringing an exception condition to

A version of this paper was presented at the Second ACM Symposium on Principles of Programming Languages, Palo Alto, Calif., Jan. 20–22, 1975.

This work was supported in part by contract DAAA25-74-C0469, Frankford Arsenal, Philadelphia, Pa.

Author's address: SofTech, Inc., 460 Totten Pond Road, Waltham, MA 02154.