SOFTWARE—PRACTICE AND EXPERIENCE, VOL. 11, 733–740 (1981)

The Consequences of One's First Programming Language

RICHARD L. WEXELBLAT
Sperry Univac, Blue Bell, PA 19424, U.S.A.

SUMMARY

After seeing many programs written in one language, but with the style and structure of another, I conducted an informal survey to determine whether this phenomenon was widespread and to determine how much a programmer's first programming language might be responsible.

This paper discusses my observations, gives informal results of the survey and suggests the need for a more formal study.

KEY WORDS Programming languages Teaching programming Education

BACKGROUND

What experienced programmer has not seen programs written in one language that have the style of another language? Having experienced 'FORTRAN with semicolons' and 'C with a BASIC flavour' over the years, I began to wonder whether the programmer's first programming language had an effect on programming ability as profound as the effect of one's native language on one's thought patterns.

Referring to what has been called the Sapir–Whorf Hypothesis, the logician W. V. O. Quine has said 'One frequently hears it urged that deep differences of language carry with them ultimate differences in the way one thinks, or looks upon the world'.[1] Dwight Bolinger characterizes the hypothesis:

> Since every language has a form and no two forms are the same, it follows that no two cultures having different languages can have identical views of the world. Instead of a perfectly flexible rubber mask that shapes itself to reality, each language is somewhat like a Greek mask with its built-in scowl or grin.[2]

Over many years of programming, teaching programming, and debugging other people's programs, it has seemed to me that something akin to the Sapir–Whorf Hypothesis applies to programmers—especially to those who have never been taught to abstract the development of algorithms from the development of programs. Indeed, I began to worry about the consequences of loosing on the computer centres and computer science departments of the world a horde of programmers whose total background is a rudimenatry BASIC learned in elementary or high school from a teacher who has virtually no actual programming experience.

Kathy Hirsh-Pasek, who read an earlier version of this paper commented that I '... should realize that the Whorfian hypothesis has both a strong version, "language

0038–0644/81/070733–08$01.00
© 1981 by John Wiley & Sons, Ltd.
Received 14 August 1980
Revised 3 November 1980

determines thought", and a weak version, "language influences thought". The strong version which is depicted by this quote (i.e. Bolinger), is almost always contested. The weaker version is virtually tautological'. My feeling is somewhere in between.

Of course, there are many fine teachers of programming at the pre-college level—and my rude remarks also apply to some who teach programming outside of the high schools and elementary schools. I see two interrelated problems: the teaching of the *programming language* and the teaching of *programming per se*. Although there are some who can ably and successfully teach the former, there are lamentably few who can teach the latter. It is the effect of poor teaching and limited languages on the malleable minds of youth that concerns me here.

If the language does even influence our way of thinking about programming then it is quite important that we teach the concepts of programming before we teach the language. Otherwise the biases will become ingrained.

In 1978 I published a request provocatively entitled 'Youthful indiscretions, or is BASIC bad for your health?' in a series of journals, asking for information on experiences that might confirm or deny my hypothesis. (The text of the letter is appended below).

I received almost 50 replies broken down roughly as:

30 addressing my points
12 not relevant
4 pointers to other sources
2 attacking me for raising the issue in the first place.

SUMMARY OF FINDINGS

There seems to have been no serious attempt at a controlled statistical study of the later effects of the first programming language. Indeed, the problems of designing such an experiment would require a major commitment of time and resources and a very large population of students studied for years. I have come to this conclusion after thinking about the enormous number of independent variables and their cross-correlations. For any observed aspect of programming behaviour, there seem to be dozens of equally valid alternative explanations. Without access to a large sample of students learning the same language on the same operating system from the same teacher, etc., there seems no way to isolate cause and effect from noise and artifact. Hirsh-Pasek adds that this is why psychology and education are separate disciplines. 'You can't do pure research and control variables in the classroom. Even if teacher, book and socioeconomic status are controlled, individual and other differences wash out results'.

With the understanding that no universal truths are implied, here are some highly subjective conclusions. There is no hard evidence to support or to deny any of these opinions.

- Programmers tend to favour their first language even to the extent of applying the style or structures of this language when they program in other languages.

  This tendency may be reduced by many other factors such as the length of time the first language was used exclusively, the skill with which the second language was taught, and the innate ability of the programmer.

- Those who begin with an assembly language often have an unfortunate tendency to prefer assembly language. Otherwise there seems to be no long-term effect on the ability to assimilate new high level languages.

  I believe this to be due to the very low level of most assembly languages. The programmer is forced to abstract the conceptualization of the problem from the coding of the solution.
- The effects of poor teaching of programming methodology are probably as important as the effects of a poor first language. Although (I repeat) there is no significant quantitative evidence for this point of view, many programmers, teachers, and managers of programmers have independently come to this conclusion.

The usual result of linguistic myopia (or poor training) is the inability to abstract data and control structures, the inability to separate the problem solving from the solution implementation. In current jargon, the problem relates to the differentiation between procedural and functional abstraction. A common manifestation is the tendency to go directly from problem statement to code without any intervening design steps. When a problem is too big to code directly, the programmer may try to cut off small chunks and code them in the hope that 'getting something running' will eventually lead to a solution. Such a *modus operandi* is typical of the bottom-up design methodology that is, I hope, discredited as an overall design strategy.

Programmers who think of code in concrete rather than abstract terms limit themselves to the style of data and control structures of their ingrained programming habits. A FORTRAN programmer who has successfully designed a payroll system may not see any value in the ability to combine numeric and alphabetic information into a single data structure. BASIC programmers often find no use at all for subroutine formal parameters and local variables.

The mind-set problem, works both ways: the PASCAL programmer who is forced to use FORTRAN or BASIC may end up writing awful programs because of the difficulty in switching from high-level to low-level constructs. APL converts seem to abhor all other languages.

TABULATION OF RESPONSES

My initial query was too loosely worded. After looking at the responses, I tried to formulate a series of statements and then to tabulate (or in some cases deduce) agreement or disagreement.

0. The first language does indeed affect later programming ability.
   Yes: 12, no: 5 (Most of the yesses are from people observing other people; the noes, all from people describing themselves.)
1. Those who start with a limited language may find it hard to make full use of higher level languages later on.
   Yes: 7, no: 7, maybe: 2
2. BASIC is a limited language.
   Yes: 5
3. A small language is a good first language.
   Yes: 9, no: 1

4. For beginners, the distinction between writing a program, translating it and executing it is missing.
   Yes: 2, yes (but good): 3
5. The programming process is not learned when a limited language is used first.
   Yes: 3
6. Good first languages are:
   PASCAL: 5
   PL/C (or similar): 5
   Algol: 2
   PL/I: 2
   Assembler: 2
   FORTRAN: 1
   APL: 1
   Structured pseudo-code: 2
7. Bad first languages are:
   COBOL: 7
   FORTRAN: 3
   PL/I: 3
   Assembler: 2
   BASIC: 2
   Algol: 2
   APL: 1
   SNOBOL: 1
8. Good later languages are:
   PASCAL: 2
   LISP, Assembler, Algol, APL: 1 each
9. Bad later languages are:
   APL, BASIC, COBOL: 1 each

Obviously, not all respondents addressed all of the above items. I was interested to note that the majority (5 of 7) answering no to item 1 themselves began programming in machine or assembly language.

Some of the respondents' comments are interesting:

- Prior experience with FORTRAN and PASCAL interfered with learning LISP.
- Learning a 'strange'first language (APL, SNOBOL, LISP) may limit later ability to program in a 'classical' language.
- Assembly language programmers are overly concerned with efficiency.
- COBOL programmers have no insight into programming. (This, from someone who professed to be a long-time COBOL programmer who had begun programming in assembly language).
- Managers who know only BASIC cannot understand the need for software engineering.
- An advantage to learning BASIC first is that you can avoid the need to worry about data structures.

One respondent was a student about to take her first programming course. She was concerned that learning BASIC would harm her ability to program. (I encouraged her

to take the BASIC course but to keep an open mind and to seek out other languages to learn and use.)

Another respondent declared that he began programming with BASIC and was not now limited in programming ability in any way. Therefore he is the counter-example that disproves my hypothesis.

At the 1980 ACM Small Computer Conference, I spoke on this topic. Several in the audience approached me afterwards with related experiences and anecdotes. All those who took a programmer's viewpoint commented that my hypothesis is true enough and applies to many people—but not to the speakers themselves.

OTHER COMMENTS

Several respondents provided insight into related problems. I will quote from some of these.

Sam Calvin sent me an interesting report at least part of which has been published in the AEDS Monitor.[3]

He feels that in a computer-illiterate society, the decisions that concern us all will be made by a computer-literate few. This underscores the need for good training from the start.

Ned Chapin who has been in the computer field for more than 25 years observes that if a person has a 'flexible turn of mind and learns easily' then it does not matter what the first language is. Starting with a low power language will slow the development but have no long-term effect. For the majority of people, however, the view of the world imparted by the first language will only slowly be broken. In his own book,[4] Chapin treats data structures in one chapter and algorithms in another, independent of any particular language or formalization.

John Nicholls, whose book on programming languages I heartily recommend[5] comments:

> My observation is that, more important than one's first programming language is one's first substantial (i.e. 6 months to write) program. At one time I had a theory that many people after completing their first program spent much of their remaining professional lives trying to repeat it, improve on it, or prove it is the best. This comes from discussions with a number of very experienced programmers and I have found it enlightening, while trying to understand the seemingly irrational behaviour of a colleague, to try and understand their early, formative experiences.

Alan Perlis, who successfully made the transition from Algol to APL, has stated:[6]

> 'I've heard from so many people who program in Algol—or PASCAL now in the universities—that they are not willing to learn APL because of the high threshold investment in their time that it takes. They are not willing to spend the time to think the way I think you have to learn to think if you're going to program in APL—to learn to think, so-to-speak, of arrays not as mathematical objects but as carriers . . . .'

Since the most exciting (to me) breakthrough in computer accessibility is the advent of the home or personal computer, the following comment by Don Tarbell is relevant: 'Perhaps the single worst problem that the computer hobbyist faces right now is the lack of a really good high-level language'.[7] Although Tarbell recognizes the need for improved handling of labels, names, I/O, and control structures, it is interesting and significant that he does not mention any requirement for improved data structures.

INTERPOLATION

It should be obvious that the appeal of BASIC is its ready availability in low-cost systems in an *interpretive* implementation. In an article on programming in the home of the future, Bruce Anderson[8] argues that programming will be a widespread activity. For effective use of resources, users must become fluent programmers and that fluency must come from source-level interaction with a real-time evaluator. He cites LISP, POP-2 and Smalltalk, but of course, at least in the U.S., BASIC is pandemic. In my opinion, PASCAL is not the answer (what is the question?) but I welcome the spread of low-cost PASCAL interpreters for minicomputers.

CONCLUSION

Although everything presented here is anecdotal evidence from which it would be irresponsible to draw firm conclusions, I cannot help add that my concern about the future generation of programmers remains. Who knows? Perhaps the advent of automatic program generators and very high level specification languages will so change the way we talk to computers that all of this will become irrelevant.

I doubt this, though, so I again encourage those who have significant results or who are in a position to design and conduct experiments to publish. And please, let us involve trained statisticians in the design and evaluation of these experiments.

Phil Dorn, an old friend who has been involved in programming for years, accused me of being a snob. He sees 'no connection between what one was first taught and ability to use the features of the newer (better?) programming languages'. Is BASIC a programming language? 'It certainly isn't a programmers language, but that's a different matter. BASIC is a reasonable tool for the casual or part-time computer user and let's leave it at that'. He adds that the teacher is more important than the language. My reply to Dorn sums up my feelings on the matter.

> . . . No I (hope I) am not being a snob. I am expressing a concern based on limited experience and observation, and trying to confirm or deny an hypothesis. All of us (old-timers) started with some form of assembler and then graduated (or regressed . . .) to higher level languages. The important thing is that we started with a language that gave us the full power of the machine and knowing what could be done, we continued to exploit fully whatever language we happened to be condemned to use.
>
> I'm not really worried about the professional computer scientist. My worry is that the hordes of children now working in subsets of BASIC will never be able to make full use of COBOL to write clear, understandable code. You see, the children are not being taught to program, to express algorithms. They are

> being taught to do quick and dirty (always dirty, never quick) run-it-once programs . . . .
>
> Things will straighten out in time, I'm sure. It's just that we are facing a generation of computational Flatlanders who will dominate the coding rooms for a decade or so.

These flatlanders will either be robots who code the same program over and over again or, worse yet, be like the character in the recent movie 'Being There', who uses all of the right terms but whose abstract understanding of them is void. If my inferences are anywhere near correct, the first language does determine the mental set for the second and other languages and we must concentrate on the teaching of programming as a concept and as a methodology not in Graduate School but from the start.

ACKNOWLEDGEMENTS

I would like to thank all those who replied to my query, especially the people whose responses I quoted above. Thanks also to Kathy Hirsh-Pasek, Mike Schneider, Jeff Schrager and Dennis Hamilton for comments and discussions based upon the first version of the paper.

APPENDIX

(Text of original inquiry)

Youthful indiscretions, or is BASIC bad for your health?

There is at least one branch of Cultural Linguistics that teaches that the deep differences in languages lead to differences in the way one looks upon the world. (Sometimes referred to as the Sapir–Whorf hypothesis, ...)

I would be interested in hearing about experiences later in life of those who have first learned a simple programming language (BASIC, LOGO, PILOT, etc.). In the world of p-baked ideas ($0 <= p <= 1$) I have an idea that is about 0·48-baked (i.e. slightly less than half-baked) and that hypothesis is that few programmers who begin their programming life with a language so limited as BASIC ever significantly extend their ability to make full use of the data and program structuring capabilities of higher level languages learned later.

In any case, I solicit from programmers, managers of programmers, and teachers of programming answers to the following questions:

What is the effect on later ablility to program of a person's initial programming language?

Is BASIC (or FORTRAN, COBOL, PL/I—you supply the name) a good or bad first language? Why?

If the programmer's first language is 'different' (e.g. SNOBOL, LISP, APL) from the common languages (e.g. Algol, FORTRAN, COBOL) what effect does this have on later programming ability?

If enough interesting or significant material appears, I hope to be able to put together a report affirming or denying the hypothesis stated above. All contributions will be acknowledged and all replying will recieve a copy of the conclusions.

(Published in *SIGPLAN Notices, Personal Computing, Software—Practice and Experience, APL Quote-Quad* and others.)

REFERENCES

1. W. V. O. Quine, *Word and Object*, Wiley, New York, 1960. p. 77.
2. D. Bolinger, *Aspects of Language*, Harcourt, Brace, Jovanovich, New York, 1975, p. 241.
3. S. Calvin and D. Robertson, 'Computers in a large school system 1966–1980', *AEDS Monitor*, **17** (4–6), 22–23 (1978).
4. N. Chapin, *Computers, a Systems Approach*, Van Nostrand, New York, 1971.
5. J. Nicholls, *The Structure and Design of Programming Languages*, Addison-Wesley, Reading, MA., 1975.
6. A. Perlis, 'Almost perfect artifacts improve only in small ways: APL is more French than English', a talk given at the APL 78 Conference in Los Altos, CA on 29 March 1978.
7. D. E. Tarbell, 'The computer hobbyist—will he be left behind?', *Proc. IEEE COMPCON.*, IEEE, New York, 1977. pp. 302–304.
8. B. Anderson, 'Programming in the home of the future', *Int. J. Man–Machine Studies*, **12**, 341–365.