

# Language Considerations

Dennis Ludwig  
Aeronautical Systems Center

*A major question asked when beginning a project is, "What programming language should I use?" This article will provide some ideas to help make this choice. First, it will present some real-world examples of how this decision has been made in the past, and then some decision-making parameters will be explored. It is intended that these ideas could be formalized into a decision table that could be the basis of a decision-making process.*

Back when the 6502 microprocessor was competing with the 8080 microprocessor, my college professor posed a question to our class. Why did IBM choose to use the Intel 8080 for its personal computer when he considered the 6502 to be a superior microprocessor?

His answer was that the IBM engineers used the 8080 because they were familiar with it. They knew how to program it, and they had to get a product out the door fast. So they used the product they knew best.

Similarly, during the time the government was under an Ada mandate, which meant all new software development would be Ada, I knew of a company that presented a proposal to develop a flight line electronic warfare tester using Fortran. Why? Their microwave engineers knew Fortran best.

Obviously, one common method of choosing a programming language for a new project is to let the engineers decide, and go with the language they already know. However, there can be problems with this approach. System design should have a team approach that considers all life-cycle phases. This includes asking, "Will the design engineers be around for the maintenance phase?" "Will the choice be made on what is best for the project, or on what is best for the engineers?"

The project manager must ensure that the language selection technique being used is not pure careerism, which is making decisions based on what is best for someone's career instead of on what is best for the organization. For example, during the era of the Ada mandate, professional magazines were listing plenty of job offerings for C++ programmers, but none for Ada programmers. Although Ada is a superior language with better array handling facilities, stronger typing, more readability, etc., marketable programmers were perceived to be the ones with C++ experience. More recently, the trend among programmers has been to push Java as the language to handle all projects. Have you checked the job listings lately?

It would be easy to simply let market demands decide which language to use. Some people feel that they cannot go wrong with the most well-known language. Their rationale is that if it were not the best, it would not be the most well known. Currently, that means using Visual C++ or C# hosted on a Microsoft Windows operating system. For many projects, this will do the job. However, a popularity contest is not the best way to make what is essentially a technical decision. A software product that works very well in one project may not be suited for another under different circumstances.

---

*"The project manager must ensure that the language selection technique being used is not pure careerism, which is making decisions based on what is best for someone's career ..."*

---

Quickly changing markets could leave a project stranded with an obsolete software base.

Another method of choosing a programming language could be called the Dilbert method. Under this scenario, executives lament that they are not able to find Ada programmers because the local schools are not teaching Ada. Meanwhile, their respective companies are advertising to hire C++ programmers, but not Ada programmers. The executives whine that they do not have any control over their advertising departments. So we are led to believe that the advertising or human relations departments make the programming language decisions. The Dilbert

rationale is that since the languages major companies advertise for in trade journals determines what students want to learn, this then determines what colleges teach. Since major companies rely on colleges to put out the latest, or at least the best, technology, they advertise for what is most popular with the colleges.

This author believes our society is currently trapped in an unhealthy cycle that is propagating an inferior language. A way to improve the process of deciding which language would be best for a particular development project should be developed and documented.

## Language Selection Process

The first step in a language selection process is to decide what computer hardware and operating system will be used. Defining the hardware and operating system greatly narrows the choice of available languages. If an Atari 800 were going to be used for the brains of a robot pet, then a compiler or assembler for that machine would be useful. If plans call for using RSX-11 on a program decision package, then that environment will bind the choice of languages.

To help in choosing an operating system, ask the following three questions: "Will direct access to the input/output (I/O) ports be necessary?" "Will multitasking be needed?" "What support will be available?" If direct control of I/O ports is needed, then an operating system like DOS or Linux or a real-time operating system that provides this service will be required. However, Windows will not allow it. If multitasking is required, then DOS drops out of consideration. If direct access to I/O ports is not required, then Windows can be considered. The chosen operating system will constrict the language choice.

Lastly, the availability of support maintenance for an operating system also factors into the final choice. Operating systems themselves are large computer programs that have bugs and will require support. Unfortunately, operating systems are numerous but few have good support.

(For more information on operating system considerations, see [1].)

The next step in the language selection process is to define or decide on a design method. Some considerations are to create a flow chart, top-down design, or object-oriented design. If an object-oriented design is required or desired, then strictly procedural languages like Fortran drop out of consideration. Some languages that claim to be object oriented are Ada 95, C++, Common Lisp, and Smalltalk. Ada 95 and C++ can also be used as procedural languages using flow charts or other design tools.

The following list itemizes some things to consider in choosing a language. Using this list, managers could build a decision matrix, assign weights, and arrive at a decision that would be documented.

1. **Speed.** It is hard to beat assembly language for speed, but some compiled languages with optimized compilers can match it. Avoid interpreted and scripting languages if possible because they are slower.
2. **Operating System.** Defining the hardware and operating system greatly narrows the choice of available languages.
3. **Program Size.** There is a difference between programming large systems and writing a small program to calculate a mortgage amortization. For large system programs, a large system programming language with clean interfacing is required. Ada would be a good choice here.
4. **Reuse and Cost.** If it has already been coded, why reinvent it? If the need is for a small neural network, buy a book with the code included. Then it can be rewritten to make it faster or more robust if needed.
5. **Engineers' Knowledge.** It takes two years to learn a language like Ada or C++. There are two-week courses and 21-day instruction books for most languages, but it takes two years of really working with the language to become good. Look at what languages are already being maintained in the organization. Some synergy could be gained from staying with what is already being used, considering any trade-offs with obsolescent factors. Or ask, "Is it time to upgrade the workers' knowledge?"
6. **Required Pointers.** Some languages like early versions of Fortran, Java, and Basic do not have pointers. Others do, including C, C++, Delphi, Ada, and the latest version of Fortran.
7. **Other Data Structure Consider-**

**ations.** Will enumerated types or records (such as struct in C) be required? Will array slicing be required or helpful? Is string manipulation built into the language, provided as a library, or will it have to be coded separately?

8. **Garbage Collection.** The reclamation of heap-allocated storage after its usefulness in a program is called garbage collection. Automatic garbage collectors can be useful for some applications, but can be damaging in other situations because it relinquishes control of the program. For some languages that do not have automatic garbage collection, like C++ and Ada, a routine could be written, if required. If you do not need it, it does not matter. If garbage collection interferes with what you need to do, then choose a language that does not have it. Not having automatic garbage collection gives the programmer more control.
9. **Reliability.** This can be one category or several such as information-hiding capabilities, readability, or strong vs.

---

*"The language decision is a fundamental design decision that will affect production, testing, training, and maintenance, so the entire system life cycle should be considered."*

---

weak typing rules. Typing strength is a matter of opinion. Every book the author has on C++ claims that it is a strongly typed language, but any language that intrinsically converts floats to integers has no business calling itself strongly typed. This is another issue to be addressed in the organizational software-engineering handbook.

10. **Standardization.** Will one company's compiler produce the same results as the compiler from another company? Will a later version of the same compiler work with earlier code? The last question has been a problem with C++ because the language has changed rapidly over the years. Will the language be around in 10 years or

more? Ada's well-documented standard and the Association for Computing Machinery (ACM) special interest group assures its success. On the other hand, the former lack of standardization for C++ impeded portability and programmer efficiency, and even though it now has a standard, it is not being followed. Furthermore, Microsoft C++ is not the same as Borland C++. And while Pascal and C are also standardized, Java is not.

11. **Compiler Tools Like Debuggers.** If a graphical user interface (GUI) is mandated or desired, that is also considered because the compiler would have to interface with the GUI tool. It would take considerable research to get honest ratings in this category. The operating system would be a major player in this consideration. If Microsoft Windows were the platform, Visual Basic, Visual C++, and Ada would be major contenders. But if Unix or Linux platforms would be users as well, then the visual languages would not be considered.
12. **Parallel Processing.** If parallel processing is required, some languages shine brighter than others do. The operating system would also be a consideration here. For most applications, the decision matrix would have *not applicable* here. Ada tasking features make it a good choice for parallel processing needs, but other languages, such as Pascal or C++, could be pressed to perform in this arena.
13. **User Base.** Be careful here. C++ is touted as the most popular programming language today, and it claims a large user base. Because Borland C++ differs from Microsoft C++, and almost every compiler sold has its own brand of C++ (due to lack of standardization), the claim to a large user base can be misleading. The backward compatibility to C has been used as an excuse to expand the user base, but only very trivial programs are actually backward compatible. The ACM has a user group for Ada, and that assures a viable user base for this language. Because Ada is well standardized, the users are truly more portable.
14. **Specialized Areas.** ATLAS is a standard language for automatic test equipment. Although most modern test stations use some form of BASIC or C, keep in mind that there are languages developed for special purposes. Forth was developed to control telescopes, but has been expanded into a

powerful, flexible language. Lisp and Prolog are languages associated with artificial intelligence. However, general-purpose languages are often used in specialized areas. For example, array slice capabilities of Ada make it a powerful language for database operations.

With these considerations and others that may be program-dependent, a payoff table or decision matrix could be built to make a decision based on program requirements. A good process would eliminate programmer bias and focus on the project. It would also provide documentation that could be used to build experience for decisions in future projects. For more information on developing decision matrixes and using weights in decision making, see [2, 3]. Note that in management jargon, a decision matrix is also known as a payoff table.

### Other Considerations

To further enhance the process, the decision maker should be familiar with some basic software engineering concepts. For example, suppose a group inaccurately argues that they want to use C++ instead of Ada because they do not think information hiding is a good idea. Information hiding is not an Ada concept, but a software engineering concept that is also used by C++ in the form of file scope and class scope. The concept is associated with Ada because this language has constructs (packages) that make implementing it easier. A small pamphlet explaining this and other software-engineering concepts would be useful for managers to dispel misconceptions.

Unfortunately, software engineering is not an exact science, but a field full of opinions, contradictions, and poorly defined words and concepts. For example, the concept of *object oriented* has changed over the years. A software-engineering handbook would help to standardize some of the concepts, even if the definitions are just accepted in your organization. It would have to be a *handbook* and not an encyclopedia, or busy program managers will never read it.

Another problem in language consideration is knowing them all well enough to make decisions. It is difficult to judge if a language will meet your requirements without knowing it. Yet no one has time to learn the syntactic structure of all of the available languages. For example, an excuse to use C on a project instead of Ada could be that "the array sizes had to be dynamically allocated, and Ada could not do that." Of course, Ada could do

that, but the person did not know that. Thus the decision would be made on incomplete or inaccurate information.

As another example, consider that Java does not have pointers. However, someone who knew the language could build a linked list using the Vector class in `java.util`. Keep in mind that insertions and deletions are less efficient (see [www.cafeaulait.org/javafaq.html](http://www.cafeaulait.org/javafaq.html) question 4.1). Language summaries similar to one found at [www.cs.rochester.edu/u/scott/pragmatics/a.html](http://www.cs.rochester.edu/u/scott/pragmatics/a.html) could be expanded and made part of the process documentation [4].

In the future, the line between operating system and language may get blurred. For example, Niklaus Wirth, the creator of Pascal, along with Jurg Gutknecht have created Oberon, which is an integrated software environment that can run on bare hardware or on top of a host operating system. Oberon is also the name of a programming language. It is the author's understanding that the operating system and the language were developed to operate closely together (see [www.oberon.ethz.ch](http://www.oberon.ethz.ch)).

### Language vs. Process

Some people feel that the process and tools are more important than the language. A common belief is that if the design is good, the language selection will not matter. This author believes that the process, tools, and design are dependent on the language. The language decision is a fundamental design decision that will affect production, testing, training, and maintenance, so the entire system life cycle should be considered.

If the popular language today is not around during the maintenance phase of the system, life-cycle costs will greatly increase. Languages that are not standardized may be around, but they might be drastically different from the original, or even from another language going by the same name (as is the case with C++). Front-end costs to move to a different language would include training and purchasing tools. Back-end costs to maintain an obsolete system would be constant training and trying to purchase or maintain obsolete tools. Jovial programmers, for instance, are hard to find. COBOL is another example, as those who fought the Year 2000 battle found out.

### Conclusion

Operating systems are complex programs. Compilers are complex programs. Choosing which combination of these complex programs best suits a need is not

an easy process, and most often, this process is not documented. This article provides an idea for making and documenting the language decision process using a decision matrix. The table elements presented are not intended to be inclusive, or to even all be necessary. The idea is to use a decision process that is documented and able to be improved. ♦

### References

1. Silberschatz, Galvin. Operating System Concepts. 6th ed. Indianapolis, IN: John Wiley & Sons, June 2001.
2. Babcock, Daniel L. Managing Engineering and Technology. Prentice-Hall, Inc, 1991.
3. Griffin, Ricky W. Management. Houghton Mifflin Company, 1984.
4. Scott, Michael L. Programming Language Pragmatics. San Francisco: Morgan Kaufmann Publishers, 15 Jan. 2000.

### Additional Reading

1. Ghezzi, Carlo, and Mehdi Jazayeri. Programming Language Concepts. John Wiley and Sons, 1987.
2. Friedman, Daniel P., Mitchell Wand, and Christopher T. Haynes. Essentials of Programming Languages. The MIT Press, 2001.
3. Jones, Richard, and Rafael D. Lins. Garbage Collection, Algorithms for Automatic Dynamic Memory Management. John Wiley & Sons, 1996.

### About the Author



**Dennis Ludwig** is a computer engineer at the Simulation and Analysis Facility, Aeronautical Systems Center at Wright-Patterson Air Force Base, Ohio. He has worked with software for more than 20 years. He has a bachelor's of science degree in electrical engineering from Louisiana Tech University, a master's of science degree in administration from Georgia College, and a master's degree in engineering from Mercer University.

ASC/HPMI  
2210 5th Street Bldg. 146  
Room 122B  
Wright-Patterson AFB, OH 45433  
Phone: (937) 255-7887  
DSN: (785) 255-7887  
E-mail: [dennis.ludwig@wpafb.af.mil](mailto:dennis.ludwig@wpafb.af.mil)