What's Wrong With Forth?

John S. James
CommuniTree Group
P.O. Box 486
Santa Cruz, CA 95061

ABSTRACT

Forth offers unusual control over the entire hardware and software environment, allowing design efficiencies far more important than its speed and memory performance. But some traditional coding practices have used this flexibility in ways which impede the development and maintenance of large software projects. And easy access to system facilities has too often allowed vendors to get away without providing complete application support for any particular purpose.

The keys to improvement are modular software design, information hiding, and closure. We will have succeeded when programmers can join a project, then quickly come up to speed and contribute within a single section of a complex system.

***

Why doesn't Forth have more "market share" compared with other tools for software development? I asked this question for several years before sensing any clear answers.

Forth today finds itself completely secure in its own niche; there is no chance it will go away. It has kept growing over the years with little institutional backing, against alternatives which have multimillion dollar development budgets, major promotion campaigns, and public subsidies going for them. Only UNIX has been a serious competitor, but it is not growing as a threat.

Yet with all the demonstrated power of Forth, it has had near zero attention from academic computer science departments, even though they have long been aware of it. And Forth remains very much out of style among those who finance or manage large software-development projects. Most users of Forth (myself included) got involved largely by accident; once involved, few persons or organizations have left Forth for anything else. Yet those who have chosen software-development systems from scratch, without first-hand experience with Forth, have rejected it from its beginnings 15 years ago through the present.

How can a system work so well without being more attractive to new users?

What's Right With Forth?

No one knows why Forth works as well as it does. The usual lists

of benefits and features -- lists heavily skewed toward raw machine efficiency, of limited importance in today's industry -- do not explain its usefulness in practice. Many languages, even BASIC, are interactive, and several are extensible.

The key benefit may be that Forth is interactive and extensible at all levels at any time. Users can combine the lowest levels of direct access to machine instructions and bits, with intermediate levels such as the words which implement the FORTH system itself, with the highest levels of application-oriented, user-defined data types. And users often do combine all these levels -- even in the same line of code.

The result: an almost unprecedented control over the entire machine and software environment, making possible system-wide design efficiencies far more important than instruction speed and memory compactness, where Forth also performs well. And all too often, another result: code which is "structured", but otherwise undisciplined, unfactored, and full of unique concoctions -- a nightmare for anyone but its author to maintain.

Large applications may have thousands of different words. Forth permits, but does not require, the organization of these words into cleanly defined modules, with precisely known, documented interactions. Without such organization, it can be hard to modify any part of the system without understanding many other parts, causing obvious problems for ongoing development and maintenance.

Eventually, Forth may evolve into a new language which includes better support for software modularity. Meanwhile we can begin this evolution in our own work by using better practices to design and implement maintainable software.

Better Software Practice: One Underlying Theme

The big question for maintainability of Forth code is how much new programmers must learn before they can do useful work on the system in question.

Unfortunately, software practices which have evolved over the years do not measure well against this standard.

Problems and Recommendations

The problem starts with the Forth system itself. The implementation styles of the systems in use have set the style for application programming.

Forth differs from other languages in that almost everyone who does much work with it becomes an expert on its internals. The fact that most users can do so, easily and gradually, provides the immense advantage of complete control of the environment, allowing optimum use of the hardware for the job at hand. But all too often, users not only

can, but also must, become experts on Forth internals to do useful work.

The designers of other language products, for example those using Basic, C, or Pascal, must know something about their audiences and provide complete facilities for some purpose, because most users will not change the compiler or the language. But few Forth systems provide complete facilities for any application purpose. (For a quick lesson on the deficiencies of your system, pick up a Basic book and start implementing all of its examples.)

Few Forth systems are ready-to-go for any application purpose, such as business, machine control, or science. Most systems sold amount to language construction kits for these purposes, not languages. Users not only can design their own language, they must.

Forth is indeed a complete language for one particular purpose -- implementing the system itself. Words like WORD, HERE, and CONVERT can of course be used in applications, but they have taken their current form largely for the convenience of system implementers, not application programmers.

Other words, such as the brackets, blur the line between application and system. That's no crime, but problems start when application programmers must understand the internals of their particular system in order to do professional-quality work. Vendors should provide a complete working environment which can be used independently of the system. Internals could still be used to extend the environment, for efficiency or to meet unusual requirements, but this system code could be kept in a small, separate section of the software.

Clarifying the boundary between system and application will also give us an occasion to review certain coding practices which don't help either our work or our reputation. For example, while unstructured jumps may occasionally be justified, should they be obtained by improper nesting of structured operations which happen to give the right result due to the particular way they were implemented? Forth practice is full of words used completely outside their designed intent to take advantage of incidental effects. And also there are minor embarassments such as adding truth flags to Ascii character values or to addresses.

Forth has all the potential to be a major contributor to the computer industry. But we must use it to provide clean development environments, so that doing useful work does not require digging into the internals of application modules, or of the system itself.