General Planning Considerations for Fortran 2000

by Craig T. Dedo
January 24, 1997

This paper contains several loosely related opinions and concerns which are related to the planning process for Fortran 2000. I delivered some of these remarks at the May 1996 X3J3 meeting. Other parts are more recent concerns which I have.

1. Paradigm Shifts / Basic Assumptions

As we prepare to develop plans for Fortran 2000, we should keep in mind the concept of the paradigm shift. Paradigm shifts are central to strategic planning in all kinds of organizations. "Paradigm shift" means that the mental models for understanding the business environment which used to work no longer work, due to technology advances, changes in customer preferences, and other changes in the environment. New paradigms are necessary. Making the transition to new paradigms is always a difficult and uncomfortable process.

We may be experiencing a paradigm shift and should keep this in mind when we design Fortran 2000. I am specifically thinking of two assumptions which may no longer be valid.

1. We can safely ignore interactions with the operating system.
2. We do not need to specify anything about processor character sets.

In both cases, our previous assumptions may no longer be valid.

There may be other assumptions which are no longer valid. We need to critically examine the core assumptions we have about how the world works and what the market values.

2. Planning Horizon

Please keep in mind that we are designing a language for the years 2001-2012. We need to anticipate the environment that Fortran will be in and the needs that users of Fortran will have during that time.

3. Strategic Direction / Marketing Considerations

Although some persons may disagree, we need to be aware of and respond to marketing and strategic direction considerations as well as technical issues. The often-expressed attitude of, "We are a technical committee, not a marketing committee" simply will not work in any organization which has strategic direction responsibilities.

I am very concerned about the image which Fortran has and the continuing lack of popularity of Fortran, particularly in the academic community. Six years after the release of the Fortran 90 standard, many persons still consider Fortran to be an archaic dinosaur with little relevance to the world of modern computing. In many places in the academic world the teachers still say, "Fortran is outdated. You don't need to learn it." While there may be little we can do directly to combat such backward attitudes, there is much that we can do to avoid aggravating the situation.

We especially need to be open to modern programming concepts and features which have proven their worth. We also should strenuously avoid mistakes like those made with Fortran 77, where the designers refused to admit into the language many modern features such as recursion, derived types, long variable names, a complete set of control constructs, and pointers.

In summary, we need to constantly ask ourselves where we want Fortran to go in the next few decades.

4. Target Markets

While I have often stated that I believe that Fortran should be a general purpose language and useful for teaching programming, I do not believe that we should abandon or neglect the needs of our core market, which is high performance numeric computation, particularly in science and engineering. One of the basic principles of strategic planning is that businesses should always act to serve the needs of their core market.

We should not give up the markets of general purpose application development and use of Fortran as a teaching language. Although C and C++ dominate these markets and there is stiff competition from Microsoft's Visual Basic in the Intel-based PC market, I believe that there is still opportunity for Fortran to capture a substantial share of these markets if we make the right decisions and provide users of Fortran with the tools that they want and need.

Another area which needs attention is giving users what they want. Very often, users have requested a standardized version of a feature for which there are differing and incompatible vendor extensions. In many of these cases, a large proportion of users consider such features to be highly desirable, necessary, or even vital. Right now, a good example is support for standardizing a command line interface. There are many other features which also fall into this category. Although many such features involve close interaction with the operating system or similar difficulties, we really should be in the business of figuring out how to satisfy the desires of our users, not making excuses why it cannot or should not be done. As the old Marshall Fields slogan goes, "Give the lady what she wants!"

One other concern is human productivity. This is simply how much software functionality, such as function points or feature points, which can be produced with a given number of direct labor hours. I believe that Fortran has a major advantage in this area, due largely to its generally straightforward and easy to understand grammar and syntax. We should build on this advantage. We can do this by accepting into Fortran 2000 powerful concepts and tools which allow the application developer to more closely model the real world, without a lot of effort and potential for mistakes. It also is desirable to design new features in a way which is easy to understand. And we can continue to eliminate gratuitous irregularities.

To summarize, I believe that the target markets which Fortran should serve should be these:

1. Robust application development in numeric, scientific and engineering applications with particular emphasis on high-performance numeric computation.
2. Be second to none as the language of choice for general purpose application development and teaching.
3. User demand, i.e., give the customers what they want.
4. Increased human productivity, i.e., software functionality per direct labor hour.

5. Value of Application Programmer Interface

I believe that as we develop new features, we need to keep in mind the value of a high quality interface for the application programmer. This means syntax constructs which are straightforward, intuitive, and easy to use. Following is a quote from the mathematician and philosopher Alfred North Whitehead on the value of a good notation:

By relieving the brain of all unnecessary work, a good notation sets it free to concentrate on more advanced problems, and in effect increases the mental power of the race. Before the introduction of the Arabic notation, multiplication was difficult, and the division even of integers called into play the highest mathematical faculties. Probably nothing in the modern world would have more astonished a Greek mathematician than to learn that ... a large proportion of the population of Western Europe could perform the operation of division for the largest numbers. This fact would have seemed to him a sheer impossibility ... Our modern power of easy reckoning with decimal fractions is the almost miraculous result of the gradual discovery of a perfect notation.

-- Alfred North Whitehead

This principle applies to computer programming languages as much as it does to any other area of human endeavor. In my experience, a lot of time and effort is wasted fighting the limitations of the application's programming language rather than solving the problems that the application was designed to solve. The development of user-friendly syntax for new features could help a lot toward making Fortran a far more productive language than previously. This, in turn, could help Fortran increase its popularity.

6. Ground Rules for Reconsideration of Previously Rejected Features

Recently, there has been some controversy on what the ground rules should be for consideration of features which were considered and rejected previously. Unfortunately, there appears to be a significant fraction of committee members who, for whatever reason, believe that once a feature has been rejected once, it should never be considered again.

I believe that such an attitude is not in the best interests of Fortran, nor in the best interests of any of the Fortran Standards Committee's constituencies. A feature may be rejected at any particular time, for a wide variety of reasons, including a heavy schedule of more important items, difficulty of implementation, or lack of significant user interest. However, in the area of programming languages, where there is rapid technological progress, conditions change dramatically from one release of the International Standard to the next. A feature which may be difficult or expensive to implement in a particular year may become much less expensive or far easier to implement by the time the next version of the standard will be ready, simply due to technological progress.

It is generally the rule, rather than the exception, that major progress in human endeavors requires several attempts before they are successful, rather than just one or two attempts. In a debate on comp.lang.fortran around January 20, 1997 on a proposal for command line arguments and environmental variables, I presented an analogy of 14 events from world history, from the time of Moses to the present, where repeated attempts were required in order to make progress. Each of these events had a major impact on the quality of life today. In responding to criticism of my analogy, Glen Clark (Glen Clark, posting on comp.lang.fortran, dated 22 Jan 1997) made the following observations:

> If we're going to resort to straw dogs, we're never going to get anywhere. Craig Dedo was not trying to seriously discuss biblical history and everyone who read it knows that. It was an illustration called an allegory which makes a point by drawing a parallel. I thought he made his point well.
>
> The position of the Committee appears to be that once an issue has been decided against, that is the way it should stand for eternity. And that anyone so brash as to suggest a

second iteration of discussion is a pariah. Craig's point was that repeated attempts is not the exception, but rather is the rule for many successful human endeavors.

I think the belief that many have is that there should be some kind of feedback loop between the users and the Committee. While being mindful that committee members are people too and that their time is not unlimited for the committee, there is a sense that there should be some semblance of an iterative process, even if the number of iterations is low. If feedback is neither welcome not possible, what is the purpose of a DRAFT STANDARD? In the absence of an iterative feedback process, the final will not differ from the draft, so why call it a draft?

Further, there is tremendous legal liability in being part of an "industry standards group". Federal antitrust laws formally require a very visible and responsive procedure. For more information, see "US vs Hydrolevel". I don't have a citation handy, but any patent or IP attorney will know it by heart from the case title.

[End of quotation from Glen Clark]

It is my hope that committee members will take his words to heart and develop a sound process for reconsidering previously rejected features and develop closer communications with the general public.