The Future of Software Design

Industry looks to software as the source of the next wave of innovation in microcomputers

by William Gates

Software, after years of taking a backseat to hardware, has finally come into its own. Today there is general acknowledgment of software's importance. It is the bridge between the machine and the user—the tool that brings the power of the computer to the user. And software is defining today's crucial information issues.

Instead of the emphasis of past years on building better and more powerful machines, the emphasis now is on how to harness the full power of the existing hardware through improved software design. The promise is that the existing machines could do the job much better—more easily, more efficiently—if software were better designed.

And this promise, in turn, leads straight into several key issues that are facing software developers today. What, exactly, constitutes a better design? Of the various approaches that software design can take, which will be most effective in helping users access the full potential of their machines?

Currently software developers face five major issues. None has easy answers. The stand that each of the major players in the field chooses to take on these issues—and the degree to which the ultimate judge, the user marketplace, accepts each stand—will determine the direction of software design.

A great deal of money will be invested in these choices. The cost of developing a fully integrated family of applications is enormous. Apple talks of investing $50 million to develop a complete applications family; Xerox views the job in terms of hundreds of man-years. Therefore, each software developer is going to have to take a good hard look at each of these issues and make its choice with great care. A wrong choice will be costly at best; at worst, it could spell financial disaster.

In this article, I'll examine today's central software issues, analyze the pros and cons of the possible choices within each issue, and hazard some guesses as to which directions will prove to hold the key to the software packages of the future.

This month the BYTE West Coast editors relinquish their forum to Bill Gates. As chairman of the board of Microsoft Corporation, Gates directs the efforts of one of the microcomputer industry's major software houses and has some definite opinions about the arrival of the soft revolution.

Integration

Integration has been a byword in the software industry for some time. But the issue here is not superficial integration. I am not talking about taking various products and calling them by similar names. I am not even talking about moving the data back and forth between the products through some sort of low-level numeric description, where special commands must be given each time the user wants to move data from one application to another.

Such an approach, although *better* than no integration at all, presents the user with two major problems. First, special commands take considerable time and effort, both in the initial learning and in their application each time the data is to be moved. Worse yet, with this type of integration, important information about the data is lost. Take sales data, for example. In a particular application, users may have described sales by time period (daily, weekly, or monthly), by sales unit (sales rep, product line, or division), and by the form in which they want to print it. With today's level of integration, if they try to move this data from one application to another, they generally will lose some of these important descriptors. The data will be devoid of its full structure.

The two key features of real integration, then, are that it must capture all data descriptors and it must be automatic. That is, to get two applications to work together, there should be no need to continually move the data back and forth manually. If, for example, users need to combine data from their balance sheets and their income statements to do monthly reports, they should be able to specify what data they want the reports to include and in what format

it should be printed. The rest should be automatic—graphs, charts, and all—without any need to go back in and reinput or redescribe the data.

This is how fully integrated software will work. But the big question is, how do you get there? Basically, two possible approaches exist: either build one single application that does everything or else find better ways of moving data between separate applications.

The first approach has a definite appeal, in view of the fact that no one has yet developed a way of moving data between applications in a high-level form. But there are three significant drawbacks to the idea of building a single applications package that does it all.

First, there is the problem of specialized expertise. Even if one software developer had the expertise to build a complete set of generic applications—time scheduling, project scheduling, database development, electronic spreadsheets, and the like—it would be impossible to find a single vendor who had the expertise to build all the necessary vertical applications. And vertical packages specific to different professions or companies are going to be a major segment of the software market. This need, then, points to the importance of developing an approach to integration that lets different parties with specialized types of expertise come in and provide specific vertical applications of the various packages.

A second problem with the approach of developing a single application that does it all is that it requires the selection of a single data structure. Because a data structure that is ideal for one application may be clumsy and inefficient for another, the net effect of this approach is that it compromises individual applications. For example, an in-memory data structure that is well suited to a spreadsheet application may be poorly suited for a database package. In fact, it may be totally unusable. If users want to develop graphs from the data stored in all the separate cells of a spreadsheet, for example, and they have to move the cells around and give a special set of commands each time they need graphs drawn (or, alternatively, find a macro string that will accomplish the same end), they are not going to be likely to use the application very frequently. Clearly, different applications require different data structures to make them easy to use.

The third difficulty with the single-application approach is that the command structure could easily become overstrained. The number of different commands and decision trees could become a significant problem.

For all of the above reasons, Apple and Microsoft are in agreement that the best solution is to have multiple products that can easily pass data back and forth. This doesn't mean that the products cannot be priced as a single package, or that they can't all be on the screen at one time. But it does mean that they will be based on different data structures and will use different command structures.

User Interface

A second crucial decision area facing software developers today involves the development of standards for user interfaces. Developers are in general agreement on some of these issues. For example, it is generally accepted that packages should include online "help" files so that users can immediately call up a piece of help text that is designed for the specific context in which they find themselves. Similarly, menus written in standard English and full-sentence prompts are generally accepted. Visicorp, for example, is moving away from the use of coded commands (/) and toward the use of English words.

The big issue today in the area of user interfaces is the introduction of graphics. To many people, graphics implies the drawing of bar charts, isometric charts, etc. But the graphics issue is, in reality, far broader than that.

The question is how to present data on the screen. So far, companies have been fairly confined in how they use the screen to present data. For a long time, they could only put characters (and monospaced ones, at that) in specific positions on the screen. This may not seem like a problem at first glance. But stop and think for a minute: if every time you went to use a piece of paper or a chalkboard you had to take little letters and place them where you wanted them, wouldn't you find this approach to be restricting? You might find yourself using the paper or chalkboard a great deal less than you now do, when you have the freedom to put arbitrary images there in any form.

The new graphic technology, with its use of pixels and bit-mapping, is bringing this same richness to the computer screen. The ability to view the screen as a piece of paper and to put arbitrary images on it means that graphics are going to be used for a great deal more than just drawing graphs. Icons, for example, tell the user what is happening in a much more compact and compelling way than words. Cursor displays to show users their positions are another form of visual feedback. For example, when users are deleting something, the screen could show scissors moving around the material being deleted. Even graphs and diagrams will be revolutionized by the new graphics technology because the time and effort required to produce them will be significantly reduced. In fact, what the new graphics technology represents is a revolution in user interfaces.

The bottom line is that graphics are going to be a standard part of all computers. No machine that costs more than $1000 will be without a built-in bit-map graphics screen. And the software analog of that hardware statement is that, one year from today, no decent application software family, no decent language family, and no decent operating system will be without extremely high level support for this type of graphics capability. It will be no small task for the software developers to achieve this graphics integration, but it is a necessary task. Furthermore, the graphics capability is not going to be in the form of add-on packages that users go out and buy after they have bought their computers: it will be part of the definition of the machines themselves. As such, it will require

very high level primitives to allow the user to easily access the graphics capabilities.

As the above observations indicate, software developers are going to have to agree on some user-interface standards to allow the full power of this graphics revolution to be felt. First, they will need to develop some standards for incorporating the graphics capability into the machine. Apple is already moving in this direction with its development of a strong operating system as a foundation for such built-in features. Second, they will need to agree on some high-level operating system commands to make the graphics capabilities readily accessible to the user.

Data-Storage Metaphors

Selection of the most appropriate data-storage metaphor is one of the toughest issues facing the software industry today. Basically, this term refers to the way the user perceives the storage of data within the system. Take Apple's Lisa system, for example, which is supposed to be capable of being learned in 20 minutes. Learning the spreadsheet application is going to be easy only for people who are used to working with formulas—people who like formulas, who understand them, and who understand how they can work together in an interdependent fashion. A data-storage metaphor that is based on placing formulas in cells of a spreadsheet is never going to be easy for most people to learn, regardless of how the system is dressed up with easy-to-remember icons, simple English commands, and so forth.

Xerox, on the other hand, uses a linear, document-oriented metaphor. It includes different types of frames (text, graphics, and so forth), but the orientation is still that of a document, which is scrolled through in linear fashion.

The direction that Microsoft is taking is toward a database metaphor. We undertook a study within our own offices to look at the ways people ask about and record data. Our findings showed that the data itself is the key; people generally take a database approach in recording and accessing information. Someone wanting sales figures for the previous year, for example, would not create a spreadsheet with empty cells and then send it to the accounting department to have the cells filled in. Rather, the person would start with the data that he had and request the additional data needed to complete the picture.

You can see that the metaphor question is entirely separate from concepts such as graphic icons or windows. It is also a much more difficult issue to deal with. The effort, however, will definitely be worth our while: it is in this area, more than any other, that we can make the breakthroughs that will allow the ordinary user to view the computer as simple. A software approach built around the right metaphor will allow users to walk up to the machine, immediately see the data that they have put into the system, and then easily choose the applications that will allow them to view that data in the formats they need—all without having to refer to files, spreadsheet cells, formulas, or any other complex constructs.

Tying Personal Computers to Mainframes

A fourth major concern that software developers need to address is the growing interest in tying personal computers into mainframes. Because of the significant differences among mainframes, this is no simple matter. Mainframes—even those made by the same vendor—have different file handlers, different communications software, and different operating systems. The IBM 370 alone has at least six major operating environments and, within each of those, multiple databases. Creating the software that will allow a personal computer to tie into such a machine will not be a trivial task.

The problem is not simply tying two machines together. That has already been done: software exists that will turn the personal computer into a terminal, ignoring its local intelligence.

The difficulty is to create a method of tying the two together that will allow automatic database querying. Users should not, for example, have to know JCL (job-control language) to access data from the mainframe. Nor should they need to learn a complex set of command structures. Rather they should be able to query the computer for data anywhere in the system and have the system itself use its intelligence to retrieve that data. In fact, the way the data was initially described in the dictionary should tell the system where to go to get it—whether to go, for example, to the mainframe, Compuserve, or Dow Jones. Resolving this software problem will not be easy, but it must be accomplished; the increasing use of personal computers in large organizations makes this a central concern today.

Expanded Definition of an Operating System

An important development that you will be seeing in the near future is a greatly expanded definition of an operating system. Microsoft, for example, as the vendor of one of today's most popular operating systems, MS-DOS, is planning to incorporate an increasingly higher number of functions into that system. Graphics capabilities, user-interface capabilities, networking—all will be incorporated into the operating system. Instead of these functions being considered add-on products, they will automatically be a part of every machine. This means that applications writers will be able to assume that these functions are there and design their packages accordingly.

The Soft World Is Here

As the above observations indicate, the innovation taking place in the world of computers today originates with software. No longer do you need to go out and build better, more powerful hardware to achieve productivity improvements: you simply develop a new software package, and people can put it to use immediately in their existing machines. The revolution is here—and it is soft.■

William Gates is chairman of the board of the Microsoft Corporation (10700 Northup Way, Bellevue, WA 98004).