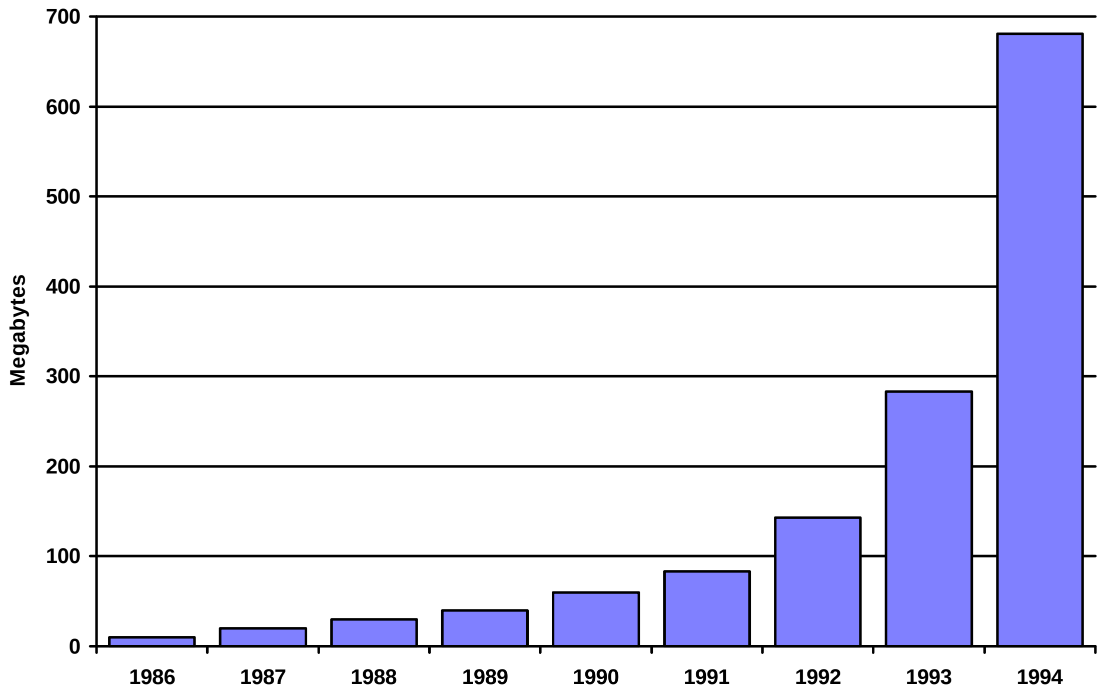
Software Bloat - Is it Here to Stay? *1994 by Herb Chong*

Have you ever wondered how all that space on your hard disk is used ? Doesn't it seem like yesterday that a 10M hard disk, or any hard disk at all, was an incredible amount of storage that would take a long time to use up? I went through my old Byte magazines to research this article and tallied up the hard disk sizes in some of the systems reviewed and previewed. Combining that with some data published in PC Week and a little black magic, I ended up with this chart. It purports to show the average hard disk size shipped with the "average" system today. The 1994 figure is PC Week's estimate for the end of 1994. Extraordinary -- isn't it?

If you take the numbers and do a little arithmetic, as I did, you will find that the average disk size almost doubled each year from 1986 to about 1990, and then more than doubling since then. I don't think it is any coincidence that Microsoft Windows 3.0 hit the market place in May of that year. People have more applications on their systems if they are Windows users than if they are DOS users and those applications are bigger. Windows applications tend to come with more features and are just generally bigger than their DOS counterparts.



Is there an end in sight to this rapid growth? To answer this question, we need to look at some of the reasons for the rapid software size growth these past several years, what causes lie behind those reasons, and finally, what assumptions they create about how people use their systems.

More - Cheaper - Faster -, and Sooner!

Take a look at the Windows applications on your computer. Have you upgraded any of them since you got your first version? What has changed from version to version? Each upgrade promises that you simply cannot do without the new features that the older version doesn't do. The packages are getting skinnier, especially if you get the CD-ROM version. They seem to come out at an ever increasing rate on ever increasingly tight delivery schedules. It's a trend that started in the '80s and is continuing. Let's look at these and other factors and how they influence software size.

Staying Even with the Competition

It seems that the only real justification for an upgrade is to get new features that you want. Marketing's job is to convince you that you really need these features. Otherwise, they are not going to make any more money from you. A software vendor in the PC world doesn't sell you a subscription*yet*, they make a one-time transaction. To keep in business, a software vendor must continue to sell to new customers. What better way to get new customers than to convert all your old customers to new ones by obsoleting everything they own? If they are going to make people pay for their software, they have to convince people that they need something they don't already have.

Just in case you have any doubt, the marketing department spends a lot of time and money convincing you why the latest features are ones you really need to have and what new things you are going to be able to do with their new version that you can't do with the old. There is no doubt in my mind that all new features are useful. The real questions are how useful they are and to how many people? As the software market and consumer sophistication mature, it's harder and harder to find genuinely useful features for a large portion of the users.

Nonetheless, if you decide that one or two features are sufficiently useful to you to upgrade, you'll upgrade to get them. When you do, you get all the features you don't need as well. The programmers spent time writing and debugging the code, and the code ends up on the installation disks and your hard disk. You pay for them all. The software vendors and the programmers will argue that the cost of adding all these features isn't a lot more than adding some of them, and this will allow them to satisfy more people than they would otherwise be able to. No doubt this is true, but there is a fine line between adding a feature simply for the sake of adding a feature and adding a feature because many users can't do without it.

Let’s use Word for Windows as an example. I use Word for Windows the most of all the applications on my desktop. One of the handiest features to come along in Version 6 is the AutoCorrect feature. If I forget to hold down the shift key when I begin a sentence, it capitalizes it for me when I press the space bar. If I forget to hold down the shift key in the middle of the sentence when I press the “i” key, and then space, it upper cases it for me. If I hold down the shift key too long and the first two letters of a word are capitalized, it lower cases the second letter. It remembers that I type “don;t” frequently when I really mean “don’t” and fixes the mistake. Autocorrect is a really useful feature because I’m a self-taught touch typist and I have picked up some bad habits.

Again using Word as an example, I have yet to find someone who prefers to move text by using drag-and-drop instead of cut and paste, either via the keyboard, toolbar or menus. It’s harder to position the cursor for an exact paste and so people frequently drop the text in the wrong place. I know that it took someone a some nontrivial amount of time to get it working...and it does what it is supposed to do. How many people really benefit from it? Not nearly as many as Microsoft hoped when they introduced Word for Windows Version 2.0 and highlighted this as one of the most significant new features.

There’s the competition too. After Lotus introduced SmartIcons into its Ami Pro word processor and received favorable press, Microsoft and WordPerfect had to follow suit, whether or not it fit into their style of working. As soon as one of the big three word processors introduces a new feature into their program, it becomes a point of comparison between the programs. Adding features becomes a game of marketing and programming one-upmanship to come up with new features for these programs. The features themselves makes the competition play catch-up and allows the program to reach out to yet more of the users who might otherwise choose something different. Every extra line on the features comparison chart cost you more money and disk space, whether you use it or not.

RTFM (Read The Fine Manual)

Have you noticed that manuals are getting thinner and thinner? I have. As I upgrade my one hundred or so Windows applications on my main computer, I manage to find more and more shelf space to put the third party books I have to keep on buying to understand something that isn’t in the manual anymore. That shelf space comes from the new version’s manuals occupying less space than the ones the old version occupied. I somehow manage to net out at about the same amount of space as I used to.

The information that used to be in the manual has to go somewhere. If you are willing to live with slow access times and keeping the right CD-ROM in your drive at all times (I’ll ignore those of you with jukeboxes), you need any extra

disk space for the on-line versions of the manual. If you don't want to do this or don't have a CD-ROM drive, you have to put the manuals onto the hard disk. Yes, it's nice to be able to look up things from wherever you are, but how many of you actually prefer the on-line manuals to the paper ones? There are too many things that just aren't easily suited to on-line use. This includes tutorials and detailed reference information.

I ordered an upgrade from Microsoft Visual C++ 1.0 to 1.5 recently. It only comes on CD-ROM media and doesn't come with manuals. You need to pay $100 for the manual set. This is a continuing trend in Windows software distribution. Hardcopy manuals have been shrinking and shrinking. The information formerly in hardcopy is being shifted to on-line documentation because it's cheaper for the vendor. In this day and age of increasing competition and ever diminishing profit margins, trading a $30 manual for a few $1 diskettes is something that can't be ignored. Reducing the cost of the software is somewhat offset by the extra disk space for the floppy disks and the space taken up on the hard disk.

Guess who has to pay for the exchange of disk space for manuals? Marketing has always managed to sell or at least confuse the issue by concentrating on the great things you can do with on-line help like hypertext and searching, that you can't do with a hardcopy manual. Frankly, the Windows Help Engine isn't anything to brag about. I can do a few things with the on-line help that I can't do with the manual. There are also, a lot of things I can't do, like reading it without turning on the computer, or having to use low resolution text and graphics instead of phototypeset output, or being able to mark it up with notes and little drawings. On-line help is great when I can't carry the manuals with me, but when I'm in my office surrounded by my bookshelves, on-line help is annoying if it's all I have. I rely on the Visual Basic On-line Help because the manual is too thin to be helpful. It keeps referring me to the on-line help for the real answers *and* I get to pay for this privilege.

I Want It Yesterday !

The average Windows program is much more complicated than the average DOS program trying to do the same thing. The event driven model of application interaction places a heavy burden on the application programmer to take care of all sorts of details about making their application run. A few years ago, object-oriented class libraries and C++ became the next great thing in Windows programming. Some people went to a lot of trouble packaging up all the details and providing defaults for everything so that unless you, the programmer, wanted something different from the default, you didn't have to write anything. The class library took care of everything. Programmer productivity shot up. What used to take a year to design and write now took a couple of months.

Marketing folks went nuts. Now they could promise even more to their customers and still have a good chance of delivering.

With such pressure from all sides, programmers really haven't got much choice. They have to use development tools that let them get as much correct function as possible with as little effort as possible. *Everybody else is using them* . The tools, however, have a major drawback: they are profligate in their use of memory and disk space. People used to complain that the Windows equivalent of the famous "hello, world" program took up 20K of memory, which in the DOS equivalent would occupy a measly 800 bytes. Yet a program that takes up 10 times that much space barely rates a blink, because that is what C++ class libraries like Microsoft Foundation Classes and Object Windows Library impose on the programmers. Turn on debugging and then you see disk and memory requirements grow by another factor of five.

It's all part of how the C++ language and the Intel object format are defined. Whenever a programmer references a variable, it has to be included as part of the program whether it is used or not. The compiler can't even try to tell until you bring everything together at linking time whether something might or might not be used. In the days of C programming, it wasn't so bad because all the various variables a program could use were split across many header files and a programmer could be selective about which ones they used. That helped cut down on the number of referenced but not used variables in a program. With C++, whenever you use a class library, you have to include the entire class hierarchy every time. Doing otherwise is extremely error prone and just plain inelegant. Declaring a variable of a type in the leaf of the class hierarchy brings in everything above it right up to the top, - all their member variables and all their member functions! In the case of MFC and OWL, this can be a total of several hundred for every variable a programmer declares in their program.

When a linker processes object files to produce an executable, it knows something about which functions and external variables are used throughout. C and C++, however, do not permit the linker to eliminate unused code. Partly it is because of how C and C++ allow you to abuse the language and cause references to such objects outside of the compiler's knowledge, and partly because the Intel OBJ format doesn't store enough information for the linker to unambiguously tell if a function is really unused or not.

There isn't much choice but to leave them in. Borland thought this was enough of a problem to invent an extension to the OBJ format to allow the linker to know for sure whether something was needed or not and eliminate redundant code. So Borland Pascal for Windows programs using the same OWL class library can come in at 2/3 to 1/2 of the size of C++ programs using OWL. Do you see a stampede toward using Borland Pascal as the standard Windows development tool? Most developers don't seem to care Most can't afford to care.

Productivity is what they are measured on. Once again, you pay extra for the programmer's productivity. Unfortunately, the programmer doesn't benefit from what you pay.

It Works! What More Do You Want?

Imagine you are a new programmer on a project. The program you will be working on has been around for about three years. Remember this is Windows and C++, not COBOL. There have been four programmers before you who have worked on the code. They are no longer working on it because they have been promoted or moved on to other things. Your job is to take the list of features the team leader has negotiated hard with the marketing folks about and turn your part of that list into something that works. There's no documentation, - one hundred thousands lines of code, - and no-one to ask!

Do you dare take out any code? After all, it works now, more or less. Much safer to work in this bit here, work in that bit there, and generally change something only after you are absolutely sure of how it works. During testing, you find that sometimes garbage appears in your input. If you fiddle with it a bit, the program doesn't crash, and things seem to keep on working. All your predecessors except the original programmer probably did the same thing.

Any Windows program that has been around for more than a version or two is going to become harder and harder to add features to. First of all, new the features are more and more pervasive and more and more complex. They just can't be hacked in an afternoon. Second, adding these features stretch the original design more and more, frequently pushing it in directions that were never intended or deliberately avoided. Programs more than a few versions old quickly become frightful patchworks of elegance and ragged code right next to each other. It becomes harder and harder to enhance.

Put another way, programmer productivity is not as high as it should be. With today's deadlines for software delivery, especially in the Windows software arena, delays in delivery are very unhealthy. The faster a company can deliver new releases, the more money they make and the happier the shareholders are. It doesn't leave much room for tuning, redesign, and other such things that refine the way a program works inside. If it's not visible to the user, it's not a feature. Features sell. Taking that long pause to re-architect for the future means no new releases for a while. No releases means no income. Guess where management wants you to spend your time?

When Will It End?

Just how far can these trends continue? Remember reading about how cars were made in the '50's? Each year, there seemed to be a different bump or lump

(some people called them fins) on a car. This year's lump was in and last year's lump was out. It kind of came to an abrupt halt in the mid 60's. People suddenly wised up. Cars weren't really all that different from year to year. It was marketing of features that didn't really have much to do with what people wanted in a car.

I think that we are in a situation with Windows software where there are so many people new to software and using tools when they really don't know much about computers yet. They are swayed by the advertising and press that new versions of programs receive in review after review. When most people are able take a serious, educated look at what they do and what they need in software, I think that software sales are going to drop off.

Corporations are slower to adopt new version software because they spend more time defining the real costs of software. They understand that the real costs includes payment for upgrades they don't need, advertising to convinces them that they can't do without some feature or another, or that they will be left behind by an implicit warning against obsolescence without some wonderful upgrade or another. They know they will pay again because after the upgrade, they won't have enough room for all the other software that they need, or enough CPU to run that essential piece of software, and never enough colors to bring those games truly to life.

When consumers get fed up with being led around by the nose by the major software vendors, we'll see the rate of growth in computing power, RAM and hard disk space slow. Until then, we're going to continue to make everyone in the business richer.

Herb Chong has been a contributing writer for Windows Sources, is a Contributing writer for The Cobb Group's Inside Microsoft Windows; and is the Contributing Editor of WindoWatch.

NEXT