
How Bill Gates Redefined Application Development

(2008-06-24) - Contributed by Peter Coffee

Gates drove Microsoft's focus toward developer tools and application foundations in a way that tracked the evolving nature of the developer community.

When Bill Gates first got interested in computers, being a user of a small computer meant being a hobbyist programmer—not a gamer, not a video producer, not a social networker. It's therefore no surprise that Gates drove Microsoft's focus toward developer tools and application foundations in a way that tracked the evolving nature of the developer community.

- When anyone buying a PC was at least somewhat interested in programming, typing BASICA at the DOS prompt opened the door to an out-of-the-box ability for the machine to learn and to follow new instructions.
- When "power users" became important to the adoption and spread of PCs as workplace tools, mechanisms such as DOS batch files and command shells were there to pave the way toward building more automated environments.
- When graphical interaction moved beyond the novelty of the Macintosh—handicapped in its early years by costly and quirky development tools—to become the expected norm for mainstream applications, Microsoft's Visual Basic was an enormous leap in the ease of designing an interface and populating it with application behavior.

As a major side effect, though, VB arguably warped a generation of budding programming talent in the process—and thereby hangs a tale.

{mospagebreak=Making the wrong things easy}

In 1975, computer scientist Niklaus Wirth famously stated that "Algorithms + Data Structures = Programs" (the title of his book that is still considered part of developer canon).

Prior to the May 1991 release of VB 1.0, application development typically began with coding the program's algorithms and connecting them to data sources; the developer concluded by wrapping a user interface around the resulting application engine. Making algorithms correct and efficient and making data access robust and secure were visible and critical elements of the development process.

VB turned that process around: It encouraged the developer to begin by storyboarding the user experience, then going behind the curtain to define the results of the user's points and clicks. As soon as the program looked good and handled expected input correctly, there was a powerful temptation to declare victory and move on.

In the process, features with substantial and dangerous power were unleashed without safety mechanisms. Subsequent versions of VB, for example, introduced facilities for writing "mail-enabled" applications that could generate and send e-mail messages without the user's intervention—or knowledge.

During an early demo of VB's mail-enabling features, product testers at what was then PC Week Labs (now eWeek Labs) asked if the Windows environment itself gave the user a top-level control-tower view—not to mention a master on-off switch—for actions such as sending out an e-mail, hands-free. The response from Microsoft's representatives was, essentially, "No. Why?"

As it turned out, there were excellent reasons why safety mechanisms of this kind would have been a good idea. For Microsoft, 1999 may have ended with a whimper, as the government's protracted antitrust suit against the company produced no significant results, but it began with a bang in the form of the Melissa worm in the month of March.

Melissa's victims ruefully discovered that it was not only the independent VB developers who had access to e-mail channels and other top-level interfaces. Using the extensive automation facilities of Microsoft's Office applications, the simple but effective Melissa malware spread itself by means of electronic mail. Another notable behavior of Melissa was that it used a standard Office API to disable the security warnings of macro execution.

The fact that Melissa could do these things says a lot about the developer-centric thinking that often seems to have

shaped Microsoft's technology model.

As this writer noted at the time:

`<blockquote>`Bill Gates told us what he had in mind, 11 years ago, in a Byte magazine article entitled "Beyond Macro Programming." In his own words, recalling that article in a March 1998 retrospective, he "called for the creation of object models that would enable developers to control all the elements of an application, providing full application programmability." ...

[Microsoft] Office dissolves the boundary between operating system and application, and even between application and data: An Office document can retrieve data from remote sources, manipulate data with external code and even manipulate other applications with no intentional action by a reader (who may have no idea that these actions are taking place). ...

Windows begins with the model of a single user on a single, personally controlled machine, and never recovers from the resulting assumption that no piece of code would be on a machine if the user didn't want it to be there.`</blockquote>`

As Microsoft continued to move forward into the era of ubiquitous, always-on connection, its heritage of single-system thinking—and Bill Gates' eagerness to make things possible, often before figuring out how to make them safe—would continue to afflict the company and its customers.

{mospagebreak=The courtship of the developers}

In the time immediately prior to Microsoft's landmark release of VB 1.0, PC Week Labs analysts were briefly in possession of a remarkable document: a galley proof of the documentation for that breakthrough development tool.

What made that manual especially interesting was its liberal use of three different icons, sprinkled throughout its margins. One denoted features and behaviors specific to Microsoft Windows; the other two, respectively, marked features and behaviors for OS/2 and for dual-platform development.

It's clear that for at least some period of time, long enough to write those detailed instructions, someone at Microsoft thought of VB as a transitional tool that would introduce developers to writing for a graphical user interface—and then taking those new GUI skills to the company's future OS/2 platform.

Instead, VB was released as a Windows-only tool, and OS/2 might as well have curled up and died right then. It's almost impossible to overstate the impact of VB in triggering a Cambrian explosion of applications that were absolutely, positively Windows applications: At no time since then has it been possible for any mass-market platform to look a customer in the eye without at least some kind of answer to the question, "Will it run my Windows apps?"

The VB maneuver, which deserves to be called brilliant, put Microsoft permanently on a course of using superior development tools at affordable prices to court the developer community. It helped when competitors such as Sun Microsystems made life more than easy for Microsoft. Sun's eagerness, for example, to engage Microsoft in putting Java on Windows led to a license agreement that made it completely legitimate for Microsoft to add Windows-specific enhancements to Java as long as it fully disclosed those enhancements back to Sun. Sure, why not?

Further, Microsoft's Visual J++ was a well-designed, well-supported Java development tool—more responsive and functionally richer than anything being offered by Sun—that was perfectly capable of producing completely portable Java applications. To do so, however, a developer had to take the time to disable the useful extensions to Java that made applications easier to write and more attractive to use on Windows.

Moreover, Windows was where the vast majority of users running those applications would be, so developers had little motivation to de-feature their code or burden their development process in that way. Java's potential to lower Windows'

“applications barrier to entry,” as the antitrust court famously called it, was substantially offset.

{mospagebreak=What lies ahead}

It takes a certain sort of idealist to find any cause for complaint in the history of Microsoft’s relationship with developers. The overwhelming market share of Windows has made it attractive for developers to exploit that full-featured platform in great depth—more so than would have been feasible if there were several such platforms, any of them representing too large a market to be ignored. The low cost and high productivity of Windows development tools have yielded a cornucopia of useful applications.

It’s clear, however, that what’s worked for Microsoft in the past will be less effective in the future.

When applications run on a platform in the cloud—such as any of the platforms being offered by Amazon, Google, Salesforce.com or literally dozens of others—then the user stops seeing an advantage in running the most popular client-side operating environment. Any standards-conformant browser, running on any client operating system, becomes as good an application delivery tool as any other.

The vision of application development that Microsoft promoted before the release of the Vista update to Windows relied on a three-part promise of superior communications, superior interactive graphics and a superior model of client-side storage. Delivering that combination, known at one time as the “three pillars of Longhorn,” would have encouraged developers to think in terms of a rich Windows-specific application that exploited the client-side platform to define and deliver the user experience—while calling out to the Internet cloud for data and background services.

That was a powerful vision, but it was not delivered in anything like the time frame promised, nor were all of the pieces put together to complete the picture that had been so attractively drawn. The focus of developers has therefore shifted into the cloud itself, sometimes combined with client-side cross-platform logic using mechanisms such as AJAX—and decoupling application access from client-side platform choice.

The world in which Microsoft will compete for developer mind share going forward is therefore fundamentally different from the one in which Gates’ superb developer courtship tactics have succeeded in the past. The fully connected, always-on multiuser world of the cloud is one in which security, scalability and robustness will have to be rock solid— and Microsoft’s sheer size and the complexity of the solutions it hopes to offer to the market will represent major challenges of their own.

Peter Coffee, former technology editor of eWeek and one of the founding members of PC Week Labs, wrote reviews and columns on development tools and practices in PC Tech Journal, PC Week and eWeek from 1988 through 2007. His coverage ranged from Ada and APL to Smalltalk and XLisp, with milestones including the first published review of Visual Basic 1.0 and intensive reviews of many Java development tools, including Microsoft’s Visual J++. Coffee is currently the director of platform research at Salesforce.com.