What Netscape Learned from Cr c Software Develo p

Its development strategy produced unexpected costs, a wrong turn with Java, performance compromises, and questions about future ties to Sun Microsystems.

Netscape Communications Corp. was established in April 1994 by Jim Clark, the founder of Silicon Graphics, Inc., and Marc Andreessen, a recent graduate of the University of Illinois where he had led the team of hacker programmers that built Mosaic, the first mass-market browser for the Web. Together they founded Netscape to create a simple, universal interface that would allow users with almost any type of communications device to access the Web. Their initial focus was on two products: a commercial-grade browser that would take up where the buggy Mosaic left off, and a Web server, the software that allows individuals and companies to create Web sites [1].

Navigator 1.0, released in December 1994 as Netscape's first product, was a spectacular success, quickly becoming the browser of choice for Internet users. By December 1995, the company was worth more than $7 billion in terms of market capitalization. It soon introduced a series of browser and server products that used Internet protocols as the basis for intranets, extranets, and other business applications. Netscape thus evolved from a browser company into an enterprise software company, distinguished by its ability to write Internet software for all major personal computer platforms, as well as for Unix. By 1998, after deciding to give away the browser for free, most of Netscape's revenues were from servers, about 60% of which was from customers running various versions of Unix. The other 40% of its server revenues was from cus-

RANDALL ENOS

Michael A. Cusumano
and David B. Yoffie

R OSS-PLATFORM
D PMENT

tomers running Microsoft's Windows NT.

In 1997, Netscape encountered serious business problems. After peaking at close to 90% in early 1996, Netscape's browser share began to erode after Microsoft bundled its Internet Explorer browser in Windows 95. By late 1997, Netscape's browser, then with no more than 50% of the market, was falling steadily. In November 1998, Netscape management agreed reluctantly to a $4.3 billion takeover by America Online, Inc., which also simultaneously entered into a $1.25 billion agreement with Sun Microsystems, Inc., to help market Netscape's software and manage its software divisions. The three-way deal was approved by the U.S. Department of Justice in March 1999.

As we look back, Netscape's most significant source of leverage against Microsoft probably came from its investment in cross-platform design and programming techniques (see the sidebar "Netscape Design Techniques"). Cross-platform development was central to Netscape's strategy to become the premier producer of Internet software. The Internet now made it possible for different types of computers to talk to each other, regardless of their operating system platforms. In July 1997, Aleks Totic, a programmer on Netscape's client product division's original development team, summarized his company's position, saying, "The advantage is that your product truly works cross-platform without rewriting. We are able to release on all platforms, and it pretty much works the same."[1]

[1]This, and subsequent citations and quoted material, are derived from interviews the authors conducted with Totic and other Netscape managers and employees in 1997 and 1998.

In reality, Netscape struggled to make cross-platform development work as advertised. Engineers had to write more platform-specific code over time to remain competitive with Windows products, once Microsoft released versions of its browser for the Macintosh, Unix, and older Windows platforms. Microsoft rewrote and optimized most of its code for these products from scratch, rather than using cross-platform programming techniques. Netscape's commitment to releasing its products for all platforms quickly—by using cross-platform development techniques as much as possible and focusing increasingly on servers for the Unix market—were the key elements differentiating Netscape from Microsoft in Internet software. Nonetheless, the decision to rely on cross-platform code posed technical challenges, including unanticipated time required for development and testing and potentially weak product performance compared to platform-specific products. We detail here how Netscape tackled the challenges of cross-platform development, struggling with the various trade-offs along the way.

Cross-Platform Challenge

Ideally, there are two main ways to create cross-platform products: develop separate platform-specific versions of the product for each operating system, writing most of the code from scratch each time, and develop the bulk of the product in generic, cross-platform code, with little or no code tailored to different incompatible platforms. Netscape preferred the second approach—writing cross-platform code—although over time, it adopted a strategy combining both approaches.

Netscape engineers found that doing cross-platform development well requires minimizing several costs, or "penalties." One is the additional time and human effort needed to create abstracted, cross-platform code. A second involves tailoring at least some code for different platforms—which is almost always necessary. And a third comes from testing and debugging, as engineers spend extra time making sure features work properly on different platforms.

Tailoring even small amounts of code to specific platforms can create a logistics nightmare, because the different teams and code bases have to be syn-

Netscape Design Techniques

Despite Netscape's decision to write increasing amounts of platform-specific code, it did succeed in refining cross-platform design techniques to a considerable degree. After interviewing several Netscape programmers, we created the following summary of their most important techniques:

- Avoid platform-specific APIs as much as possible.
- Create and maintain a set of APIs representing an acceptable lowest-common-denominator interface across different platforms. That is, create a layer of programming interface abstractions, such as the NSPR layer, discussed in the main text, then write application code that connects to this layer and not to individual operating systems. Another option is to create a layer of conditional statements or instructions that tell the system what to do if the operating system is, say, X, instead of Y.
- Create a set of cross-platform components connecting to this common-denominator layer so different product groups can share them. This sharing enables Netscape's various development groups to leverage the company's investment in successful cross-platform code.
- Use as much as possible cross-platform programming languages, such as standard versions of C and C++, as well as inherently cross-platform Internet programming languages, such as HTML, Java, and JavaScript. Using cross-platform languages means avoiding such platform-specific languages as Assembler and tailored versions of such languages as C and C++.
- Keep different platform code versions synchronized by compiling, or "building," the code components daily on multiple platforms, rather than by "porting" code in the traditional sense. This synchronization minimizes different code bases, as well as the porting of code, which usually requires time for rewriting and debugging.
- Keep feature sets common across the different platform versions. Ensuring a common design minimizes creation of different code bases and tailoring work, while supporting the appearance of a common look and feel for the user.
- Tailor to particular platforms the components essential to achieving competitive performance levels on the most commercially important platforms. That is, allow teams to make at least some changes so products do not perform poorly relative to the competition.
- Do not try to write cross-platform components that developers cannot abstract easily and that might adversely affect users. Try to shift the design of these components gradually over to cross-platform approaches. An example is user-interface components unique to particular platforms, such as Macintosh toolbars, which often differ noticeably from their Windows counterparts. Over time, however, engineers can try to introduce more generic user interface components across different products, writing them in a cross-platform Internet language, such as JavaScript and even HTML. C

chronized. Keeping track of all the variations, while making sure engineers test all versions and changes properly, is no simple task. However, minimizing platform-specific code through cross-platform techniques involves its own problems. Designing products to be truly cross-platform means developers have to write code that does not incorporate any interfaces or programming "tricks" specific to a particular operating system or hardware platform. They must use relatively simple or low-level programming conventions and interfaces that are common across the different platforms. But many existing platform-specific application programming interfaces (APIs) and programming conventions enable programmers to write code that runs faster or handles graphics and memory better than code that uses lowest-common-denominator interfaces. As a result, cross-platform products can take more time, as well as result in weaker functional performance.

Microsoft engineers experienced these trade-offs years before Netscape even existed, generally choosing not to create cross-platform designs. In the 1980s, Microsoft developed Word and some other applications in a neutral pseudo-code ("p-code") format, then compiled it for Windows and Macintosh platforms. This programming method saved time in development, but the code tended to be slow. For the user, Word for Macintosh also appeared different from native Macintosh applications, since Microsoft tended to favor the Windows user-interface format. For Internet Explorer, which Microsoft initially built only for Windows 95, the company chose to create mostly separate code bases for the Macintosh and Unix versions and share only some portions of the code between the Windows 3.1 and the Windows 95/NT/98 versions.

Components and Processes

Netscape invested in components and processes to ease cross-platform development. Components included the Netscape Portable Runtime (NSPR) layer, which was used by both Netscape's client and server development groups. The server teams also shared some HTML, Java, and JavaScript components, some of the core Web server and Directory server and security code, and other common libraries for handling protocols used in more than one type of server.

A Netscape product development manager, Bill Turpin, who was vice president for server product development in August 1997, estimated that about 20% of the code in Netscape's nine server offerings at the time consisted of these packaged cross-platform components shared among the different products. He outlined the structure of the shared code in the servers, saying, "Down at the bottom level, there are things called the Netscape Portable Runtime [layer] … Those are just like operating system abstractions for file, print, socket I/O, threading … And above that, about half the servers are based on the Web server code base … One of the things we are doing to make them all common is we have a thing called the Admin server, which is a cut-back Web server [that handles setup for the different servers] … And so all Netscape servers … use our Admin servers, so they can present a common look and feel … So we're doing more and more to make them look the same and operate the same, even though they have different code underneath them."

> In reality, Netscape struggled to make cross-platform development work as advertised.

The philosophy behind the NSPR layer was to create a set of low-level programming interface abstractions for such tasks as memory management and threading (handling different tasks within the same application or on the same processor) that would work on all the platforms for which Netscape built products. The layer did what it was supposed to do—save developers' time.

But NSPR, by definition, was a lowest-common-denominator interface, entailing certain disadvantages in performance; it also had its own additional development costs. Netscape had to create and maintain a separate team of six developers to manage the NSPR layer, and they had a difficult job. First, they were supposed to serve the entire company, but the related technical demands were somewhat different in the client and the server divisions. Second, after Netscape began giving away the client code to outside developers in March 1998 in the so-called Mozilla release (Navigator/Communicator 5.0), the NSPR team took on the additional responsibility of keeping the layer current and available for outside developers. However, "available" did not necessarily mean "sta-

> Testing for the seven or so different versions of Unix took at least double the amount of resources than testing for one platform.

ble," or relatively bug-free. The team faced a continual problem in that operating system APIs and programming languages, such as Java and JavaScript, continued to evolve rapidly. At the same time, frequent changes in the NSPR abstractions led Netscape engineers to complain that the NSPR layer was an unstable foundation for creating new features.

On the server side, Netscape developers put even more emphasis on cross-platform techniques. The ability to run on multiple platforms—different Unix versions, as well as Windows NT—was the core of Netscape's sales pitch to customers. Its engineers managed to keep the different Unix versions pretty much the same, though they clearly optimized the design for the top two or three Unix versions in terms of Netscape sales (those from Sun, Hewlett-Packard, and Silicon Graphics). The company allowed some differences with the server versions for Windows NT to optimize performance, though basic features remained fairly common across platforms.

To keep the code synchronized across the different platforms, Netscape developers often tested the software simultaneously on two or more different machines, such as one running Windows and another running Unix. When checking-in their code to the project build teams, they were supposed to make sure it would "build" on both machines. Netscape developers also created and shared a series of conditional programming statements for handling differences in basic operations (though not server-specific features) among the Unix versions. In addition, to deal with variations, Netscape's server developers tried to keep the Windows NT version synchronized with the most popular Unix version—Solaris, from Sun—although developers had to keep the NT and Unix versions on separate code branches to keep from mixing them up. Later on, the developers made the adaptations needed to run the code on the different Unix versions.

Java Optimism, then Disappointment

By early 1997, Netscape executives and engineers had also become enthusiastic about Java. As a programming language, Java was inherently cross-platform. If Netscape could just write an entire product in Java, it would be eliminating many of the productivity penalties that came from designing and testing cross-platform code. Sun's promise with Java was "write once, run everywhere." This slogan worked in theory because developers do not write Java code to run on the APIs of a particular operating system. They write in a platform-neutral language called "byte code" and to a platform-neutral layer called a "virtual machine" (VM). Internet browsers and some other Internet software include the VM program, which translates or interprets the byte code so it can run on any operating system. It does not matter whether the machine is a Windows PC, a Macintosh, a Unix workstation, or a network computer.

One problem with Java is that it has to go through the extra step of being translated, or interpreted, so it usually runs or loads more slowly than code written directly for a particular operating system. But Java had other advantages that excited Netscape engineers. For example, it helped minimize certain programming errors and made it difficult for programmers to break certain useful rules, such as those needed to create object abstractions while not overtaxing available memory.

This enthusiasm for Java led Netscape to develop a Java VM in-house, because Sun's VM in 1997 still had many limitations. Netscape then planned to use its own Java tools and VM to rewrite the Navigator browser and possibly other components in the client. As Java, HTML, Dynamic HTML, and JavaScript evolved, Netscape engineers had hoped to use these inherently cross-platform Web languages more extensively to develop new components. In particular, they experimented with HTML and JavaScript for such multiplatform user interfaces as generic windows, dialog boxes, and features, such as the security manager module. After Andreessen talked publicly about the idea of a Java browser, the media began referring to this component as "Javagator," or a Java version of Navigator.

But serious frustration with Java began to spread in Netscape's engineering organization during late 1997 and early 1998. The immaturity of Java, JavaScript, and components being built with the Java language limited how much Netscape could use the new tech-

nology. For one thing, Java still lacked graphical tools to design good user interfaces. For these reasons in 1998, Netscape abandoned Java for building user interfaces in favor of C and C++.

In 1997, Netscape also attempted to re-architect the browser part of the client in what became the aborted Communicator 6.0 project. The project actually involved two objectives: rewrite the code in cleaner modules (to make product teams faster and more flexible by decoupling such major components as the mail client) and perform this re-architecting while rewriting the browser and other parts of Communicator in Java (to simplify cross-platform development). But Java did not work as well as Netscape engineers had expected. In early 1998, Netscape cancelled the 6.0 project and shifted the modularization work to a more incremental schedule for Communicator 4.5 (released September 1998) and the 5.0 project (final release expected by the end of 1999).

On the server side, programming remained more traditional, and there was little debate in 1997–1998 about Java, which was not ready for the heavy-duty requirements of the enterprise-server world. Tim Howes, Netscape's chief server architect and chief technical adviser in its Server Products Division, explained in August 1997, saying, "We have much stricter availability requirements. For us to use Java, it needs to be lightning fast. It needs to be incredibly stable. Our servers stay up for weeks or months at a time. Handling a directory server, for example, is hundreds, if not thousands, of operations every second. So performance is very important too."

By mid-1998, Netscape was not only deemphasizing Java, it even planned to replace existing Java implementations with C and C++. In November 1997, Andreessen, then with the title of executive vice president for products and marketing, reflected on Netscape's efforts to work with Java internally, the engineering problems that remained to make Java a heavy-duty cross-platform programming language, and where Java might still be useful, saying, "When you look at Java and the implications of Java, it depends on what you're trying to deliver. The ultimate point behind Java for most people is going to what I call 'dynamic on-demand Net-based applications,' where you're going to a Web page, not installing something off the CD-ROM. If developers want to create network-centric applications that can be run on any platform and downloaded over the Net, then Java will be successful—if it can be made to work, which is a large-scale engineering problem. But if developers don't want that, then Java is not right. But the theory is right. Are applications being built in a network-centric world? Absolutely. Do they need different levels of security? Yes. Will there be a wider array of devices on the Net? Sure. But the basic engineering work still needs to be done … Java is not yet at the performance, stability, or compatibility level that it needs to be to realize its promise of 'write once, run anywhere.' "

Penalties

Beyond Java, the cross-platform strategy had other costs. Several Netscape engineers estimated there was at least a 15%–20% human effort and time penalty in design and coding (excluding integration and system testing), based on the extra human and computing resources needed to develop cross-platform code, rather than, say, just a Windows or just a Unix version of the product. That is, developers working on multiplatform components might finish basic programming tasks at the same time as a team working on only one platform, but they might need 20% more people or time to do the same job. Still, a 15%–20% penalty probably represented a savings over having several separate programming teams for each platform, as long as the products found acceptance in the marketplace.

Moreover, Netscape managers had to worry about staffing the company's different version teams. At one point, for example, only one developer was the expert on the Hewlett-Packard version of Unix for the company's entire server division; this person became a real bottleneck. Another example was the Communicator 4.0 project, which shipped in June 1997, while the company was having trouble staffing its Macintosh and Unix teams. This lack of staff caused the Communicator team to fall behind and later put out a "point release" for the non-Windows versions.

An even greater penalty appeared to be associated with testing the different versions. A veteran Netscape engineer, Desmond Chan, who in 1995–1997 was the quality assurance manager in the proxy server group, estimated that integration and system testing for the seven or so different versions of Unix took at least double the amount of resources than that needed for testing only one platform. Testing for NT required even more human and computing resources. Overall, he felt that Netscape should have employed at least twice as many testers as it did to thoroughly test all the versions of its software.

Product Performance and Windows Dominance

By 1998, Netscape engineers had decided to optimize increasing amounts of code for the company's client and server products. It had become increasingly difficult to keep adding cross-platform code to the growing code bases and still ensure Netscape's

cross-platform products worked as well as Microsoft's Windows-specific products. To handle the demand of writing good code for the Windows environment, Netscape hired more Windows developers, who were not accustomed to thinking cross-platform nor particularly good at it, according to Rick Schell, the former senior vice president of Netscape's client product division.

For servers, much more so than for the client, cross-platform design posed a big risk to the operating speed of a product. Servers are supposed to be fast, and Netscape could not afford to have products that were noticeably slower than the competition. The company's engineers estimated that, when performing the same function, platform-specific server code written for Windows NT ran at least twice as fast as cross-platform code. Howes, the chief server architect, commented in August 1997, saying, "[Platform-specific code is] certainly in the range of twice as fast. Probably significantly more than that when you combine it with other things we're doing. And that is not to say that NT is twice as fast as Unix. That's to say that using the native Unix-oriented abstractions on NT slows you down vs. using the native NT stuff … I'm sure if we didn't have to worry about cross-platform at all, we'd get to go a little faster. But that's really the key to our business model. One of our basic strategies is we're cross-platform, and that's one of the big advantages that we have in the marketplace. And I want to point out that cross-platform for us is not strictly NT and Unix. There are Macs in there, but it's also NT 3.5, NT 4, NT 5, Windows 3.1, so we're cross-platform even within the Microsoft environment."

These problems forced Netscape to alter its design strategy. Through Navigator 3.0 (released in August 1996) and SuiteSpot 3.5 (released in February 1998), Netscape engineers tailored a relatively small part of the code to particular platforms—probably no more than 20%. The public source-code release of Communicator 5.0 (Mozilla), on the other hand, had roughly 40% platform-specific code, designed largely to optimize performance for Windows. Despite an espoused strategy of "cross-everything," Netscape developers focused their efforts on the most popular Unix version—Sun's Solaris—and no longer made all Netscape products available on every Unix platform.

Conclusion

Cross-platform development posed far more challenges than anyone in Netscape initially anticipated. Minimizing performance penalties with cross-platform designs after products had become rich in features was only the first problem. The company handled this difficulty through a set of design techniques, shared components, and platform-specific code whenever product performance demanded it. A more serious challenge was to recover from the decision to rewrite the browser part of the client in Java, which led to cancellation of the Communicator 6.0 project in early 1998.

In the future, we expect America Online's alliance with Sun will put even more pressure on Netscape's cross-platform strategy. First, Sun expects Netscape to cooperate in building a Java browser that runs on a variety of devices. This cross-platform Java browser will work only if Java—as a stable programming language—progresses beyond where it was in 1997 and 1998. Second, Sun is likely to pressure Netscape to tailor its servers to run best on Solaris, Sun's version of Unix. This pressure may backfire, however, because Netscape has made a success of being "neutral," that is, designing its server software to run on all major versions of Unix, as well as on Windows NT. Moreover, Netscape engineers have fine-tuned their code to make sure their servers perform competitively on Windows NT. This strategy made sense and still makes sense today because Netscape has had less competition for Unix servers, but it may have to change if Sun exerts pressure to optimize Netscape servers for Solaris.

Whatever happens, we have learned much from Netscape about the strengths and limitations of cross-platform development, at least for Internet software. Most important in this case, it seems that Netscape's successful business strategy still required compromises in design, such as mixing cross-platform code with tailored code to ensure its products could keep delivering competitive performance. C

References

1. Cusumano, M., and Yoffie, D. *Competing on Internet Time: Lessons from Netscape and Its Battle with Microsoft*. Free Press/Simon & Schuster, New York, 1998.

The authors thank Andrew von Nordenflycht of MIT for his help preparing this article.

Michael A. Cusumano (cusumano@mit.edu) is the Sloan Distinguished Professor of Management at the Massachusetts Institute of Technology Sloan School of Management in Cambridge, Mass. **David B. Yoffie** (dyoffie@hbs.edu) is the Max and Doris Starr Professor of International Business Administration at the Harvard Business School in Boston, Mass.