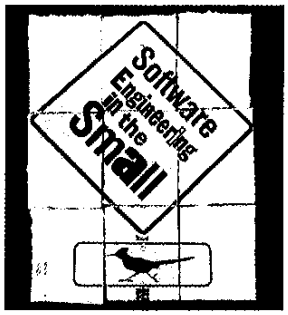
Cover Feature

Software Development on Internet Time



Product development is necessarily different in firms that must compete in fast-paced, unpredictable markets, such as Internet software. The authors reveal how Netscape and Microsoft have balanced flexibility and discipline in managing the development process.

Michael A. Cusumano
MIT Sloan School of Management

David B. Yoffie
Harvard Business School

There is probably little debate that Internet software companies must use more flexible development techniques and introduce new products faster than companies with more stable technology, established customer needs, and longer product cycles. Indeed, Internet and PC software firms have consciously departed from the sequential product development style, popular before 1990, in favor of a more flexible style. Microsoft began refining this alternative style—which we call "synchronize and stabilize"—in the late 1980s and early 1990s. The basic idea is to give programmers lots of autonomy to evolve designs iteratively but force team members to *synchronize* their work frequently and then periodically *stabilize* their design changes or feature innovations. The goal is to balance an almost hacker-type flexibility and speed with professional engineering discipline.

In 1997-1998, we interviewed more than 40 managers and engineers at Netscape and Microsoft to see how similar or different their development practices were and to see how Microsoft's Internet groups were adapting to the rapid change characteristic of Internet software markets. We found that Netscape was using a version of the Microsoft-style synchronize and stabilize process for PC software, but adapting it to build Internet browser and server products. We also found that Microsoft's Internet groups were modifying their standard process to increase development speed and flexibility.

Our interviews revealed many lessons learned using the synchronize and stabilize model and showed that this model works well for rapid, flexible software development.

This article is adapted from chapter 5, *Competing on Internet Time: Lessons from Netscape and Its Battle with Microsoft*, by Michael A. Cusumano and David B. Yoffie (Free Press/Simon & Schuster, New York, 1998).

SYNCHRONIZE AND STABILIZE: MICROSOFT

Figure 1 graphs the elements of the synchronize and stabilize model, and Table 1 summarizes how it differs from the sequential, or waterfall, approach. Basically, in the synchronize and stabilize process, developers use daily product builds, two or more project milestones, and early, frequent alpha or beta releases. Individual programmers or small teams effectively act as one large team, building large-scale software products relatively quickly and efficiently while adapting to new technologies, feature changes driven by competition, or lots of uncertainty in user requirements.[1,2]

In the waterfall model, project teams attempt to freeze a product specification, create a functional design specification, build components, and then merge the components—typically in one large integration and testing phase at the end of the project. This sequential approach was common in the early 1980s for large-scale software projects and has remained a basic model for project planning in many industries. However, it has never caught on among newer PC software companies because managers and engineers realize they can build better products if they try not to control the development process too rigidly.[3,4]

Vision statement and functional specification

Typically, at the start of a project, Microsoft teams did not try to write a complete specification that locked the developers into creating a fixed feature set. Rather, they started by creating a vision statement, usually gathering off-site to define product goals. They also worked with customer data to prioritize desired features. In more established groups, such as those developing Excel, product managers wrote the vision statement after consulting with program managers and developers. Program managers then wrote a functional product specification, which they refined as developers wrote, changed, and experimented with new fea-

0018-9162/99/$10.00 © 1999 IEEE

tures and user interfaces. The functional specification was complete only when the project ended.

Before the team started coding features, program managers and developers outlined the most important product features. The outline had to be in enough depth for project leaders to estimate schedules and organize feature teams (usually three to eight developers), often with parallel teams of "buddy" testers, usually assigned to each developer. Team members then evolved the feature set and feature details as they learned more about what should be in the product. Hence, the feature set in a Microsoft specification document might change by 30 percent or more by the project's end.

Milestones

Microsoft also tended to break the schedule into three, sometimes four, milestones, which represented completion or stabilization points for major feature clusters. Each feature team went through a complete *development subcycle*, including feature integration, testing, and problem fixing (see Figure 1). Throughout the project, feature teams or individual engineers synchronized their work by building the product and finding and fixing errors on a daily basis. At the end of a milestone subcycle, the developers stabilized the product by fixing major errors and agreeing not to change particular features or to change them only very carefully. Teams also issued alpha (internal) releases and then beta (external) releases of their evolving products at the milestone junctures. The development teams proceeded from milestone to milestone and eventually to the ship date, continuously integrating components, incorporating feedback from both external and internal users, and finding and fixing major bugs.

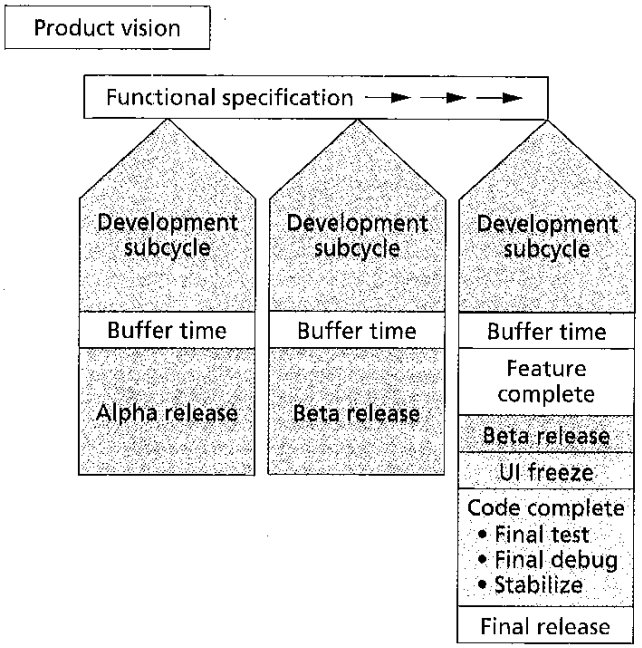


Figure 1. The synchronize-and-stabilize software development model. The project begins with the team's vision of what the product should do. From this vision, the program manager derives a rough functional specification, which the team evolves until the end of the project. The schedule has multiple stabilization points, or milestones. Three is a common number. Each represents progress after weeks of "a development subcycle": design, code, usability test, test, daily builds, debugging, integration, and stabilization. This subcycle is repeated several times, as the blue region in each milestone depicts. Each milestone marks either an alpha or beta release. The buffer time, which precedes each release, ensures that the team has flexibility to meet market demands, such as adding features that have surfaced as desirable. In the final milestone, the user interface is frozen, and the code is considered complete. The team runs a final test, debugging, and stabilizing subcycle and issues a final release. Note that only when the project is complete is the functional specification considered done. This differs significantly from the traditional waterfall (sequential) development model, which begins with a finished specification—often the road map to development.

Table 1. How the synchronize and stabilize development process contrasts to the sequential, or waterfall, approach.

Synchronize and stabilize	Waterfall
Specification, development, testing are done in parallel	Phases are completed sequentially
Vision statement is created and specification evolves (spec is output, not input)	"Complete" specification document and detailed design is done before coding
Prioritized features are built in three or four milestones	All product pieces are built simultaneously
Synchs are done frequently (daily builds), with intermediate stabilizations (milestones)	One late and large integration and test phase occurs at the project's end
Ship dates are "fixed," but there are multiple release cycles	An attempt is made to achieve feature and product perfection
Customer feedback is considered during development	Customer feedback serves as input for future projects
Large teams work like small teams regardless of project size	Many individuals work in large functional groups to scale up projects

Source: *Microsoft Secrets: How the World's Most Powerful Software Company Creates Technology, Shapes Markets, and Manages People*, M.A. Cusumano and R.W. Selby, Free Press/Simon & Schuster, New York, 1995, p. 407.

Netscape's Planning and Development Process

Step 1: Product requirements and project proposal
Advance planning meeting (APM) held to brainstorm ideas (marketing, development, executives)
Product vision generated, initially by senior engineers, now mainly by product managers
Some design and coding by engineers to explore alternative technologies or feature ideas
Product requirements document compiled by product managers, with help from developers
Informal review of this preliminary specification by engineers
Functional specification begun by engineers, sometimes with help from product managers
Schedule and budget plan compiled by marketing and engineering, and informally discussed with executives

Step 2: First executive review
Executives review product requirements document and schedule and budget proposal
Plan adjusted as necessary

Step 3: Start of development phase
Design and coding of features, architecture work as necessary
Daily integration of components as they are created and checked in (builds)
Bug lists generated and fixes initiated

Step 4: Interim executive review (if necessary)
Functional specification should be complete at this point
Midcourse corrections in specification or project resources, as necessary
Coordination issues with other products or projects discussed, as necessary
Development continues

Step 5: First internal (alpha) release (takes approximately six weeks)
Development stops temporarily
Intensive debugging and testing of existing code
Alpha release for internal feedback (or possibly a developer's release)
Development continues
User feedback incorporated
Feature-complete target (rarely met, though servers especially try to be as complete as possible)
One week to stabilize beta release

Step 6: Public beta 1 or field test 1 (takes approximately six weeks)
Repeat development and testing steps in Step 5
Server groups moving to "field tests" with limited customers rather than public betas

Step 7: Public beta 2 and 3 (each beta takes approximately six weeks)
Repeat development steps as in Step 5
UI freeze milestone
Feature-complete status "mandatory," although some minor changes still allowed

Step 8: Code complete
No more code added except to fix bugs; features are functionally complete

Step 9: Final testing and release
Final debugging and stabilization of release candidate
Certification meeting(s) with senior executives for ship decision
Release to manufacturing (RTM) and commercial release

Projects also added *buffer time* (such as 20 to 50 percent of total allotted time) at the end of each milestone so that team members could respond to unexpected difficulties or delays or add unplanned features. This buffer, together with an evolving specification, gave developers and program managers room to innovate or adapt to unforeseen competitive opportunities and threats. Most products had relatively modular architectures, which meant that teams could add features incrementally or combine them fairly easily. In the more experienced product units, managers tried to fix project resources by limiting the people allocated to any one project. They might also limit project time, especially in applications like Office, by setting specific ship dates and attaching years to product names (Office 97 and Windows 98, for example). Teams generally deleted features if they fell too far behind schedule.

Planning and reviews

Product managers usually compiled multiyear product plans so that marketing and engineering did not try to force all the features they wanted into a particular product release. Senior managers like CEO Bill Gates and other executives closely followed key projects by attending program reviews held every three months or so. In addition, project managers submitted monthly project status reports, and executives checked progress relative to three-year product plans from the divisions. Most projects had considerable independence, however, and the resources to hire as many developers and testers as they needed. This financial freedom was particularly important for new product experiments, like Internet Explorer. Ben Slivka, manager of the first three IE projects, told us that he had no budget constraint for those projects.

Microsoft continued to use its established process, first honed on products such as Excel, Word, and Windows NT, to build Internet software. The Internet groups, however, generally had shorter schedules, did more code reviews to catch bugs early, and used more external design reviews and intensive work with a select group of customers to get deeper feedback early in requirements generation and again during testing.

SYNCHRONIZE AND STABILIZE: NETSCAPE

Netscape managers and engineers had to establish a development process suitable for the time frames and uncertainty of the new Internet market. In a sense, they did this from scratch, although the company hired many people with experience at software companies such as Borland and Microsoft, and adopted heavily from what these and other companies had done.

The sidebar, "Netscape's Planning and Development Process," lists the phases (simplified) that the groups in both the client (browser) and server product divisions

generally followed from 1997 to 1998. Netscape, at least before it merged with America Online, followed the same basic pattern in process management for Internet software as Microsoft did for PC software, though again with some variations. The main differences were in functional roles and process implementation.

Functional roles and head count

In July 1998, Netscape's client product division contained approximately 240 people and the server product division about 400 people, as Table 2 shows. The numbers in the table reflect Netscape's underlying organizational philosophy: Operate as much as possible in small units and avoid adding too many people in testing. With this approach to functional hiring and team organization, Netscape was able to make the most of its relatively limited resources. In contrast, Microsoft's personal and business operating systems division, which built Windows 98, IE, and Windows NT, had more than a thousand people in 1998. Indeed, Netscape's browser division was smaller than Microsoft's product units within its divisions—Office, Windows 95/98, and Windows NT, for example—which each had roughly 300 to 450 people.

After Microsoft decided to enter the Internet arena, it gradually applied more person-hours, first to create a browser and various complementary features, and then to create new Web servers, and finally to adapt major products for the Internet. In comparison, Netscape generally had fewer people in its individual product teams. It is difficult, however, to directly compare the effort (number of engineers and time spent) Microsoft put into developing IE and Netscape put into developing Communicator or to compare each company's efforts in developing server products. Each built some components in different groups (such as security) or in parallel projects. Each carried over code from previous projects or other organizations. Different product versions and teams also overlapped in time at both companies.

Nonetheless, the number of developers working on core browser features like the HTML engine was comparable, and both companies required similar amounts of time to create similar features.[5] This suggests that productivity in Microsoft and Netscape was comparable. Netscape's performance was especially impressive because it used at least half the number of testers Microsoft used. At times, Microsoft also used larger teams of developers to catch up and add more features. Netscape took some shortcuts, however, reducing the head count by having fewer testers and by not having program managers at all.

Microsoft product units had variations, especially within the Internet groups. When acting as managers of new projects—for example, in early versions of IE and NetMeeting, which initially shipped with IE—developers usually took the lead in proposing features and writing specification outlines. Program managers came on board later and worked mainly on managing project schedules, writing up test cases with testers, interacting with interface or Web page designers, and building relationships with outside partners and customers.

Table 2. Allocation of staff in Netscape's client and server development divisions, mid-1998.

	Client products	Server products	Total
Software engineering	110	200	310
Testing (QA)	50	80	130
Product management	50	42	92
Subtotal	210	322	533
Other*	30	98	128
Total	240	420	660

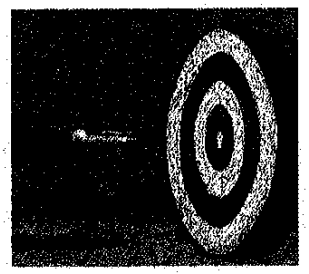
*Persons in activities such as documentation, user interface design, OEM porting, internationalization and localization, and special product support.

Product planning and reviews

Like Microsoft, Netscape evolved its process for product planning and development in stages, as it became increasingly important for projects to be more systematic and predictable. In the company's first two years, executives, such as Marc Andreessen, and senior developers drove the vision for both client and server products. Between 1996 and 1998, Netscape built up the product management staff in the product teams, which gradually became more responsible for drafting initial product requirements. At the same time, however, senior Netscape developers could form their own teams and move pet features into a new release when the technologies were ready. The developers could also exercise veto power on technical grounds. Microsoft worked the same way, with developers having veto power, although program managers generally had more technical background than product managers and were often more adept at negotiating with developers.

Advance Planning Meetings (APMs) helped bring executives, engineers, and marketers together to begin brainstorming features and release plans for each new product. Netscape divisions held APMs at least once a quarter because of the number of different products and versions. Executive reviews generally followed these meetings within a month to kick off projects more formally. After several months or so (depending on the project's length), executives held an interim review before the first public beta release to check on progress.

After drafting the product requirements document and receiving approval for a new project, developers



'The way you hit the bull's-eye the first time is by planning.'
—Netscape manager

typically shared their design ideas and preliminary design documents with colleagues and their managers as part of an informal review process. Developers sometimes created more formal design documents and held face-to-face meetings to discuss these before the midcourse executive review. How much to formalize the design documents or reviews was left up to the individual engineer or team leader. According to Jon Mittlehauser, a member of Netscape's original browser development team, any part of a feature with a user interface (UI) went through a relatively formal design and review process.

Netscape also relied on a separate UI group that put out a written spec to make sure that the user interface was the same across different platform versions. The affected groups needed to review features that touched on other features, such as the new Roaming Access feature under development in 1997-1998, which affected security and privacy components. Again, the developers responsible for the features took the lead in scheduling these meetings. (In Microsoft, program managers often coordinated requirements and scheduled reviews so developers had more time to write the code.)

The length and intensity of the planning phase varied with product complexity. For some new server products, Netscape spent as long as nine months simply generating requirements and planning the product. Servers also required extra time for cross-product coordination. In general, Netscape managers and engineers openly adopted the philosophy that, in the rapidly changing Internet world, it's impossible to specify everything that should go into a new product or a new release before coding. At the same time, they clearly believed that some planning was necessary to build complex products. David Stryker, who was in charge of building shared components from 1996 to 1997, outlined the thinking that seemed to prevail at Netscape, Microsoft, and other PC software firms:

> Here's a classic statement from a gunslinger engineer: "Do you want me to plan or do you want me to build it because, after all, if I plan it, then at the end of it you have a plan. If I build it, you have a product. Now, which do you want at the end of the process?" This is sheer b*** s***, right? The way you get places, the way you hit the bull's-eye the first time, is by planning things .... [But] you can't plan everything down to the atoms. The art of planning is articulating your goals and nongoals [what you don't want to do] really clearly and picking the things that have to be planned down to the atoms because some things do ... So we make a pretty big deal of goals and nongoals of projects. We try to make the goals measurable and concrete. The nongoals are more important than the goals because, when you're doing midcourse correction, which you do on a 72-hour basis, the best guidance you can get is to remember what you weren't trying to do.

Probably the most important midproject change to a PC software product in recent years was Microsoft's decision in April 1994 to add a browser to Windows 95. The product had already been under development since Windows 3.1 shipped in 1992. This decision dramatically changed the direction of Windows and Microsoft.

We saw some major midproject shifts at Netscape as well, such as to cancel a project to rewrite Communicator in Java modules, cancel other Java efforts, and then reorganize the client engineers into what became the 4.5 and 5.0 projects. Several managers also talked about adding Netcaster to Communicator 4.0, which shipped in June 1997, as a good example of Netscape's flexibility. This technology responded to Microsoft's Active Channels in IE 4.0. Neither feature proved successful commercially, and the Netcaster feature performed poorly in its first implementation. Nonetheless, it gave Netscape a position and some experience in the new push technology. Debby Meredith, who headed client development as well as quality initiatives in Netscape, commented in our July 1997 interview:

> Three-quarters of the way through the Communicator [4.0] development process, we decided to add the Netcaster component. It was never planned ... never in any specs. To this date, it's not in any Communicator specs because we didn't rev [revise] it. But it was a separate project on a different release schedule. It was more that we started off doing this technology, weren't really sure when it was going to hook up to one of these trains, and then just sensed that the world was right for push technology. Microsoft was thinking about similar things. We wanted to preempt them.

Product documentation

Netscape's product requirements document was usually short (five pages or so) for a new product release, at least for the client. Once engineers and executives bought off on this document in the executive review, developers, sometimes with the help of product managers, began evolving it into a rough functional specification. This spec continued to change through at least half and often two-thirds of the project, though some features (like Netcaster) might ship with the product and never make it into the written specification.

Most PC and Internet software companies will tolerate incomplete documentation because they put a premium on creating code, not documents. The Web eased the documentation process somewhat, however. Both Netscape and Microsoft created electronic documents that were extremely easy to change and structure with different layers. The new product specs were mostly HTML pages posted on internal Web sites, with indexes of features and hot links to different documents, such as API specifications. This was a major departure from pre-1994.

Project management and scheduling

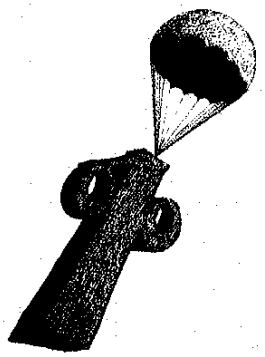
Netscape did not emphasize tight project controls. Instead, projects relied on experienced and respected engineers to use their influence over developers and act as release managers or project managers. For Navigator 1.0, this was Tom Paquin, who joined Netscape in April 1994 after working at IBM Research and Silicon Graphics. For most of the later browser versions, it was Michael Toy, who had also worked as a developer at Silicon Graphics. For servers, Netscape used directors, development managers, and test managers to oversee development in smaller product teams (called "divlets").

Generating a schedule usually began in the product marketing group, which proposed a date to introduce a new product or version. Engineers in each group tried to figure out what features they could complete in that amount of time. Marketing helped identify the highest priority items. The release then went out when teams finished or were close to finishing at least the top-priority features. Less important features could fall off the feature list—miss the train—but they often became part of the next release.

Toy tried to combine flexibility and realism with a gentle push in his schedules. Rather than give engineers deadlines, he asked developers to estimate how much code they needed to write for particular small product chunks, such as features or identifiable subsystems. He let developers adjust for a feature's complexity (how time-consuming might it be to create) and degree of self-containment (how much time might be spent coordinating with another feature group). Some engineers added a fudge factor for debugging time. Projects also added time for beta testing.

Netscape teams tended to focus not on specific target ship dates (which Microsoft managers preferred) but on three-month windows, mapping to the financial quarters. Netscape could afford to be lax about ship dates because it used the Internet for distribution and did not need two or more months of extra time to package and distribute products.

Project managers posted schedules on an intranet, rather than just sharing schedules and progress reports periodically over e-mail. Intranets provided an easy way for both managers and engineers to check on the status of the daily builds for each platform and see which modules worked and which did not. The server teams also used more formal scheduling tools than the client division and put more effort into standardizing release cycles so that Netscape could ship its SuiteSpot product set with updated versions of each server coming together at the same time. The server division had a master schedule posted on an intranet, which it called the Plan of Record. This schedule tracked the progress of the different projects and incorporated Web agents that automatically notified people of changes as they occurred.



Netscape could afford to be lax about ship dates because it used the Internet for distribution.

Component integration

At the code writing level, Netscape (like Microsoft and many other PC software producers in the 1990s) used daily builds to coordinate the work of developers, rather than relying on detailed planning and controls to supervise the work of engineers.

Check-in and change tracking. Being able to control changes is critical in fast-paced development projects. Netscape used a modified version of Concurrent Versions System, a source-code control tool freely available on the Internet. CVS was also the basis for the tool Microsoft originally used for its daily builds. CVS is not particularly advanced and has limitations for managing the concurrent development of large, complex systems, such as Communicator 4.0. Nonetheless, it was useful for a multiplatform company like Netscape because it runs on Windows, Macintosh, and Unix. Netscape engineers added their own Tinderbox and Bonsai tools to CVS to automate check-in and tracking within Communicator projects.

CVS works by comparing new and old versions of files being checked in and then creating patches for the changed code when the developer asks for an update. It then keeps track of these changes and lets the developer back up to previous files, if necessary. The Tinderbox extension was primarily an HTML page generator that showed the status of all the builds going on in the company, for all platforms, and who was checking in which components. When a build broke (files didn't compile or link properly or failed a quick acceptance test), a red box appeared, and the transgressor was likely to be the person who most recently checked into that build. Netscape also added a local newsgroup feature to post all the check-in information. Developers and testers could view this information through their browsers.

Microsoft and Netscape developers both intended to synchronize their code at least daily in the latter stages of a project.

Daily builds. Microsoft had no hard rule on how often developers should check in, but each project had to create a build at least daily. Developers who checked in code that broke the build had to fix their code immediately and face penalties, such as become the build master for the next day (who had to run tests on everybody else's code), wear a dunce cap, or pay a small fine. Consequently, Microsoft developers tended to check in and synchronize their code about twice a week on average and at least daily in the latter stages of a project, when there were lots of bugs to fix.

Netscape developers worked the same way. The client team relied on the Bonsai hook to keep track of who checked in and alert people to problems. The hook consisted of e-mail notices automatically sent to individuals who checked in code since the last successful build. Developers were in essence "on call" until someone fixed the problem. Bonsai, in combination with CVS/Tinderbox, made it possible for a relatively large team to integrate their components in just two or so hours, on average, rather than in days or weeks. In 1999, Netscape was also using this tool set to manage bug fixes and other code contributions from outside developers who were working on the public source code version of the browser, the Mozilla release. Lloyd Tabb, who took charge of building the code management tools, gave this account of the Bonsai hook:

> If you pull a build and it's not working, you can e-mail the hook and say, "Hey, what we've built today is a piece of garbage. Could anybody have broken this?" And the mail will go out to only the people who are interested parties. That list grows until the following day when the hook is cleared .... It's hard enough getting five people not to break the build. When you have 120 people, unless you're all building together, you don't ship software on time. If you have an integration step that takes a week, your cycle time is a week. Our cycle time was hours. It was a painful thing for the developers. But it also allowed us to have our ship time very short.

Daily-build testing

Although both Netscape and Microsoft used daily builds to integrate components and changes continuously, the companies differed in the way they tested the builds. Microsoft developers generally went through the following procedure, which Netscape did not have the resources to do:

- create a build and private release for themselves and their buddy testers;
- run a quick regression test (some called it a smoke test) that checked whether existing functions continued to work after adding new code; and
- have their buddy testers test the code, usually from a user's perspective.

Microsoft developers and testers normally went through the procedure before developers checked their code into the project build. The build team then ran the quick regression test again as new files came in. On at least one build per week, testers ran more extensive tests to ensure that features were working properly. Because of this extra layer of automated and manual testing, Microsoft was more likely than Netscape to catch and correct technical errors and usability problems before doing the required daily builds.

Multiplatform support. Cross-platform products provided strategic leverage for Netscape, but keeping so many versions of the code working on the daily builds was difficult. Tabb counted about 20 operating system versions on which they had to test the client. In addition, during 1997 and 1998, Netscape generally had two teams working on the same code base in parallel, such as the client 4.5 and 5.0 teams. Netscape had to use CVS to track different versions of the same product. It also used separate "branches" to build foreign language versions of its products, though separate teams handled this localization process.

Netscape groups building servers and shared components also created frequent builds—at least once a week and usually daily. But their rules and tools varied somewhat from those of the client product division. There was no daily or even weekly build of SuiteSpot. Instead, because SuiteSpot was actually a collection of separate server products (even though Netscape sold it as one product), server QA teams did spot-checking to make sure that components in the different servers worked together and with the client. A small test group in server marketing tested the common install features for SuiteSpot.

Some individual server teams preferred not to build every day because of the overhead involved. Nor did they use tools like Bonsai to keep people who might have broken the build on the hook. They did use CVS/Tinderbox to track check-ins and notify people of problems.

Code reviews. Netscape had no set policy with regard to how often or how intensively developers should inspect their code or designs. Some components received very light reviews; others received fairly intensive scrutiny. It depended on the project and the manager. Microsoft was only a little better at this: Shared components got reviewed at the design and coding phases, and most code went through at least a buddy review. But the intensity varied by project.

Slivka noted that the early IE groups used code reviews much more extensively—at each code check-in.

By the time Netscape teams were working on the last beta, the project or release manager had to approve every change, and one or two senior developers usually reviewed every code change. Netscape managers expected developers to add comments to their code, noting who reviewed it and when. For bug fixes, a tracking tool tagged each bug with a number and kept records of code changes, who checked in the code, and when.

Milestones

Milestones were important stabilization points as well as good progress indicators. For client development, Netscape's Toy focused on two targets: the first alpha (internal beta) release and the first public beta. At the project's start, the team made a rough determination of when they wanted the first public beta and worked backward to set other key milestones.

Though Netscape projects usually had a feature-complete milestone scheduled along with the first or second beta release, this milestone was not rigid. Toy explained why Netscape was so flexible with its milestones:

> You can't ship a release without features getting added, no matter how fascist you try to be about saying, "We're feature-complete, darn it!" Again, we're responding to things on the Internet. And so, suddenly, something that didn't look like it was going to be important for another six to nine months becomes important yesterday. You've got to have a response to it.

Conventional thinking about good software engineering practice says that you should have a more disciplined process and more rigid schedules so that managers can plan adequately for debugging and final testing, which enterprise customers demand. In contrast, Netscape, like Microsoft and other PC software companies, chose to let their feature sets evolve and their milestones slip because the competition, technology, and market required it. There was no point shipping a product on time if it was obsolete or contained the wrong features.

Bob Lisbonne, who headed Netscape client development after Meredith, contrasted traditional software development (along the waterfall model) to the faster-paced requirements of the Internet world:

> Traditionally, you have a complete market requirements document that's a research project in and of itself. [It's] handed off to ... developers to do a product requirements document or product specification, which is then implemented according to a bottom-up schedule, a big project-management task. That just doesn't work in the Internet world. There are too many bends in the road ... So we pursue several milestones in parallel. There's almost an alpha and a beta of the marketing requirements, and developers are working on specs well before a final marketing requirements document is done. Likewise, developers are implementing features well in advance of formal schedules being built for every piece of the product. And part of Netscape's ... nimbleness, which I think is part of its success in the market, is because we approach product development in that fashion. We are open and receptive to considering course corrections or other changes midstream.

Projects should reduce major bugs to a low and stable level before making the decision to ship.

Netscape's server groups made midproject changes in their feature sets and objectives as well, although too many major feature changes could destabilize a server product and make it difficult to sell to enterprise customers. Nor did the server groups have an alpha release milestone because Netscape's internal information systems organization would not use a new server product until it was further along in development. Instead, Netscape relied less on broad public betas and more on intensive field tests to examine the quality of the server code before the final releases.

Final-product stabilization

Shipping the final version of a software product requires tracking down, prioritizing, and fixing as many serious bugs as possible without introducing more bugs with each line of changed code. Many PC software engineers and testers, including people at Netscape and Microsoft, call this final debugging a stabilization phase. The term "stabilization" recognizes a certain reality: Given the freedom of developers to make so many late changes in a project, it is generally impossible to eliminate all bugs, but projects should reduce major bugs to a low and stable level before making the decision to ship.

In the client area, like Microsoft, Netscape allowed only two or three weeks for final stabilization. Managers relied on the daily builds, multiple beta releases, and internal QA testing to find bugs, and they expected developers to fix problems as they surfaced. The QA team also developed plans for functional testing along with the beta release plans.

Mark Tompkins was a 20-year veteran of IBM and Tandem who headed QA for Netscape's client division before moving to the application products division in May 1998. He stated that it took two to three days to test the client on one platform, such as Windows. Because they released on Unix and on the Macintosh, managers allotted a week to functional testing. After the code-complete milestone, the client division allocated two weeks of final testing—running the functional tests as well as regression tests on bug fixes to make sure

The synchronize and stabilize process works well for the Internet's very fast cycle times and high degree of uncertainty in market requirements.

the fixes worked and did not break existing functionality. They then shipped the final beta and repeated the process for the final release. Tompkins briefly outlined the steps to final stabilization:

> There's a feature-complete date that precedes ... the beta 1 schedule .... We're testing before feature-complete also, so we're testing in parallel here and our testing is causing bugs to be found .... We get to a time frame that is a couple weeks before beta, where we really tighten the screws down and we really look at every fix we've put in because this is our stabilization period .... We will continue to take fixes but we'll take fewer fixes because we don't want to destabilize .... That's our [ideal] process: feature-complete, no new features, fix as many bugs as possible, tighten down within two weeks, fix only the most significant bugs with the objective of not destabilizing it.

Bug tracking. To aid in stabilization, developers and QA engineers tracked basic bug statistics such as the number of new bugs opened, bug close rates, fix versus invalid rates (some bugs are declared invalid or "nonbugs"), won't-fix numbers, and number of bug fixes verified by QA. Most groups used the commercial Scopus tool to track bug data (it also tracked customer calls to technical support).

Netscape team leaders held bug meetings or bug counsels about once a week after they moved into the development phase. In a project's last days, groups tended to have these meetings once a day and sometimes twice a day to prioritize bugs and make decisions on what to fix. The project or release manager generally led the meetings, which brought together representatives from development, QA, and marketing.

Netscape used five bug categories—critical, major, normal, minor, and trivial. Engineers (usually from QA) who found the bugs classified them and entered them into the database. In the client division, marketing also kept a separate list of its "top 10 bugs" as reported by customers and passed this data over to QA. In the ship-decision meetings at the end of a project, which some groups called "certification meetings," QA managers went over these bug statistics and trend lines with the project manager and senior executives. They debated the product's readiness for manufacturing (creation of diskettes or CDs or certified e-copies for Web-based distribution). The groups then typically stabilized a release for one platform first (such as Windows or Unix) and later shipped the other versions within a month. Tompkins spoke about the fast pace of development:

> Speed is very, very important in this market .... twice-a-day meetings and the decisions that get made very quickly, the empowerment at the lower levels .... It almost spins your head how quick it can be.

Final release. The decision making on the final release thus combined several factors. There was usually pressure to ship as well as pressure not to ship buggy products. Compared with Microsoft, Netscape managers did not have much historical data such as expected bug rates or bug trends given certain size systems or likely problem areas in the code. And, of course, late feature additions got much less testing than features built early in the project. These problems were common in new software companies, especially those racing on Internet time to get new products and technologies to market.

The synchronize and stabilize process works well for the Internet's very fast cycle times and high degree of uncertainty in market requirements. Netscape's loose attitude toward project milestones such as feature-complete and beta deadlines had some costs: Being "slightly out of control" strained testers and the testing process, and strained technical support and customer patience as well. But this development style, aided by homegrown tools and a few rigid process rules such as daily builds and the periodic stabilizations that accompanied alpha and beta releases, also had advantages. This process gave Netscape and Microsoft an effective mechanism to coordinate large numbers of developers and testers, and it provided great flexibility in controlling even late design changes.

In comparison with Microsoft, Netscape was not unique or particularly refined in its development practices, but it was effective and efficient despite—or perhaps because of—its informality and occasional lack of discipline. Netscape's product teams were innovators in Internet technology. They also did what was required to deliver complex software products and services to new markets. They were flexible and fast as well as creative when these characteristics were important to success, and they paid more attention to quality and schedules when and where these became more important goals, such as for enterprise customers buying server suites and intranet/extranet packages. Both Netscape and Microsoft had the ability to adapt rapidly to change and to introduce more process discipline over time. These are two capabilities that we think are critical for coping with the demands of product development in any fast-paced, unpredictable marketplace.❖

Acknowledgments

We thank all the current and former engineers and managers at Netscape and Microsoft, who graciously

answered questions about development practices. For Microsoft, thanks especially to Steve Ballmer, Dave Moore, Max Morris, and Ben Slivka. For Netscape, thanks especially to Marc Andreessen, Jim Barksdale, Alex Edelstein, Julie Herendeen, Ben Horowitz, Tim Howes, Joy Lenz, Bob Lisbonne, Debby Meredith, Jon Mittlehauser, Lou Montulli, Tom Paquin, John Paul, Greg Sands, Rick Schell, David Stryker, Lloyd Tabb, Mark Tompkins, Aleks Totic, Michael Toy, and Bill Turpin.

References

1. M.A. Cusumano and R.W. Selby, *Microsoft Secrets: How the World's Most Powerful Software Company Creates Technology, Shapes Markets, and Manages People*, Free Press/Simon & Schuster, New York, 1995, pp. 187-326.
2. M.A. Cusumano and R.W. Selby, "How Microsoft Builds Software," *Comm. ACM*, June 1997, pp. 53-61.
3. V.R. Basili and A.J. Turner, "Iterative Enhancement: A Practical Technique for Software Development," *IEEE Trans. Software Eng.*, Vol. SE-1, No. 4, Dec. 1975, pp. 390-396.
4. B.W. Boehm, "A Spiral Model of Software Development and Enhancement," *Computer*, May 1988, pp. 61-72.
5. R. Verganti, A. MacCormack, and M. Iansiti, "Rapid Learning and Adaptation in Product Development: An Empirical Study of the Internet Software Industry," *Proc. 5th Int'l Product Development Management Conf.*, Politecnico di Milano, Milan, Italy, 1998, Vol. 2, pp. 1063-1080.

***Michael A. Cusumano** is the Sloan Distinguished Professor of Management at the MIT Sloan School of Management, where he teaches courses on strategy and the software business. He is the author of five books.* Competing on Internet Time *was named one of the top 10 books of 1998 by* Business Week, *and* Microsoft Secrets *has been translated into 14 languages. Cusumano consults for software producers worldwide, writes a monthly column for* Computerworld, *is chairman of the board for the* Sloan Management Review, *and sits on several corporate boards, including NetNumina Solutions (middleware software) and Marbles Inc. (real-time wireless software). He holds a PhD in Japanese management studies from Harvard University and completed a postdoctoral fellowship at the Harvard Business School in production/operations management. Contact Cusumano at cusumano@mit.edu.*

***David B. Yoffie** is the Max and Doris Starr Professor of International Business Administration and co-chair of the Competition and Strategy Department at the Harvard Business School. He is the editor of* Competing in the Age of Digital Convergence *and author or coauthor of several other books, including* Strategic Management in Information Technology *(Prentice Hall, 1994) and* Beyond Free Trade *(Harvard Business Press, 1993). He consults widely with corporations around the world in competitive strategy and international competition and serves on the Board of Directors of Intel Corp., the National Bureau of Economic Research, and several other high-tech companies. He received a PhD in political science from Stanford University.*

How to Reach *Computer*

Writers

We welcome submissions. For detailed information, write for a Contributors' Guide (computer@computer.org) or visit our Web site: http://computer.org/computer/.

News Ideas

Contact Lee Garber at l.garber@computer.org with ideas for news features or news briefs.

Products and Books

Contact Kirk Kroeker at k.kroeker@computer.org with product announcements. Contact Jason Seaborn at j.seaborn@computer.org with book announcements.

Letters to the Editor

Please provide an e-mail address or daytime phone number with your letter. Send letters to *Computer* Letters, 10662 Los Vaqueros Circle, Los Alamitos, CA 90720; fax (714) 821-4010; computer@computer.org.

On the Web

Visit http://computer.org for information about joining and getting involved with the Society and *Computer*.

Magazine Change of Address

Send change-of-address requests for magazine subscriptions to address.change@ieee.org. Make sure to specify *Computer*.

Missing or Damaged Copies

If you are missing an issue or received a damaged copy, contact membership@computer.org.

Reprint Permission

To obtain permission to reprint an article, contact William Hagen, IEEE Copyrights and Trademarks Manager, at whagen@ieee.org. To buy a reprint, send a query to computer@computer.org or a fax to (714) 821-4010.