# Product-Development Practices That Work:
# How Internet Companies Build Software

Now there is proof that the evolutionary approach to software development results in a speedier process and higher-quality products.

**Alan MacCormack**

Software is an increasingly pervasive part of the New Economy. As a result, today's general managers need to be aware of the most effective methods for developing and deploying software products and services within their organizations. Delegating such decisions to a technical staff, however skilled, can be a risky strategy. A study completed last year contains a surprising insight for managers: Dealing with the software revolution requires a process that is not revolutionary but evolutionary.

Evidence of the increasing importance of software abounds. In the United States alone, sales of software products and services exceeded $140 billion during 1998, a gain of more than 17% from the previous year.[1] In 2000, the software industry's contribution to the U.S. economy was expected to surpass that of the auto industry and overtake all other manufacturing industry groups for the first time.[2] Employment in software-related positions is growing, too. In 1998, the U.S. software industry directly employed more than 800,000 people, with an average salary twice the national figure.[3] More than 2 million people are now employed as software programmers, showing that software is not developed at a Microsoft or an Oracle but within the information-technology departments of large, traditional organizations.[4]

Software also is playing a larger role in the content delivered to customers in many industries. Nowadays, the average family sedan or high-end coffee maker may contain more software than the first Apollo spacecraft. What's more, the software features in those products may be the most critical differentiating factors. And even in industries in which software is not yet part of the products, it is playing a greater role in the products' development. As companies adopt new computer-aided design technologies, the development processes for many products increasingly resemble those found in the software industry.

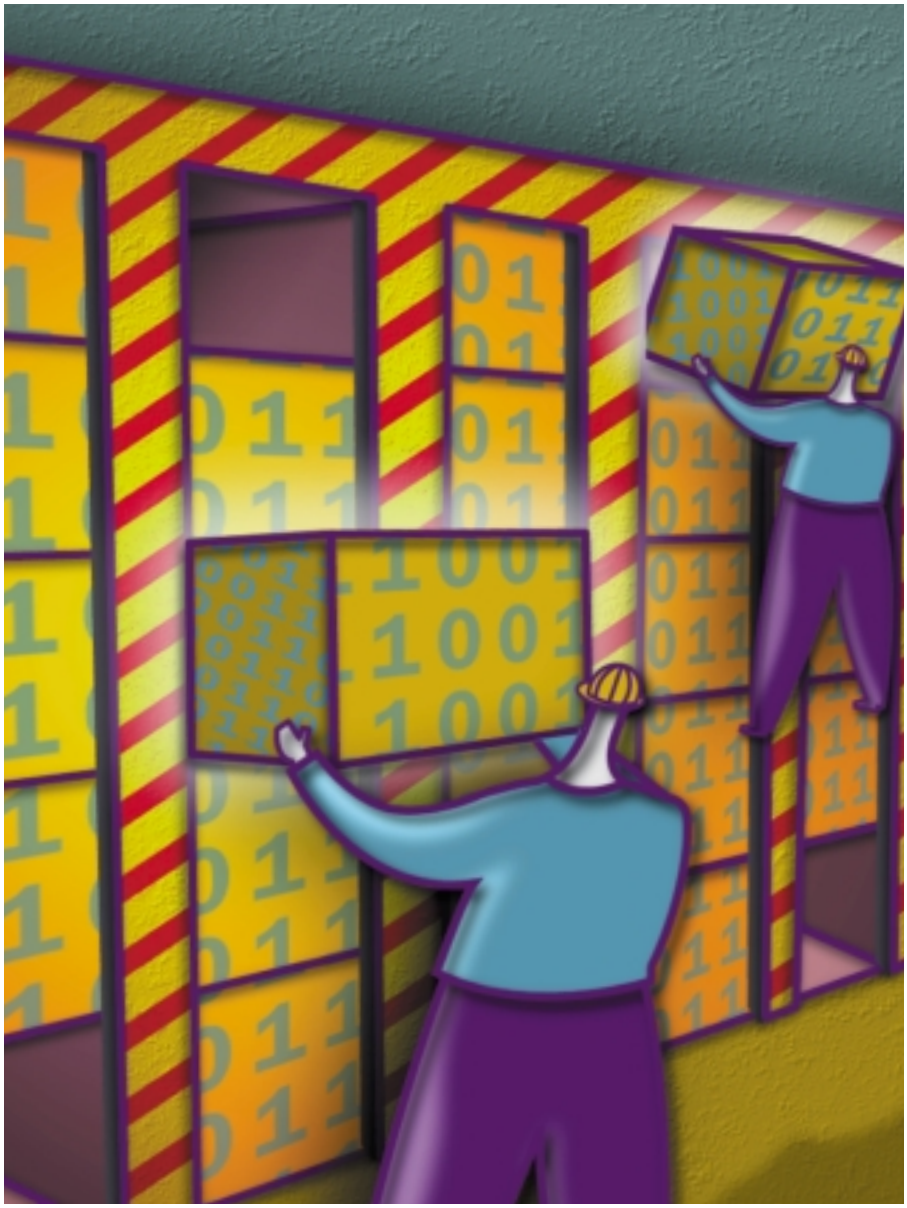
## Developing Products on Internet Time

Given the importance of software, the lack of research on the best ways to manage its development is surprising. Many different models have been proposed since the much-cited waterfall model emerged more than 30 years ago. Unfortunately, few studies have confirmed empirically the benefits of the newer models. The most widely quoted references report lessons from only a few successful projects.[5]

---

***Alan MacCormack** is a professor of technology and operations management at Harvard Business School in Boston. Contact him at amaccormack@hbs.edu.*

Illustration: Eric Mueller/Laughing Stock



## Four Software-Development Practices That Spell Success

Analysis of Internet-software-development projects in a recent study uncovered successful practices

- An early release of the evolving product design to customers
- Daily incorporation of new software code and rapid feedback on design changes
- A team with broad-based experience of shipping multiple projects
- Major investments in the design of the product architecture

Now a two-year empirical study, which the author and colleagues Marco Iansiti and Roberto Verganti completed last year, reveals thought-provoking information from the Internet-software industry — an industry in which the need for a responsive development process has never been greater.[6] The researchers analyzed data from 29 completed projects and identified the characteristics most associated with the best outcomes. (See "Four Software-Development Practices That Spell Success.") Successful development was evolutionary in nature. Companies first would release a low-functionality version of a product to selected customers at a very early stage of development. Thereafter work would proceed in an iterative fashion, with the design allowed to evolve in response to the customers' feedback. The approach contrasts with traditional models of software development and their more sequential processes. Although the evolutionary model has been around for several years, this is the first time the connection has been demonstrated between the practices that support the model and the quality of the resulting product.

### Microsoft Meets the Challenge — Internet Explorer 3.0

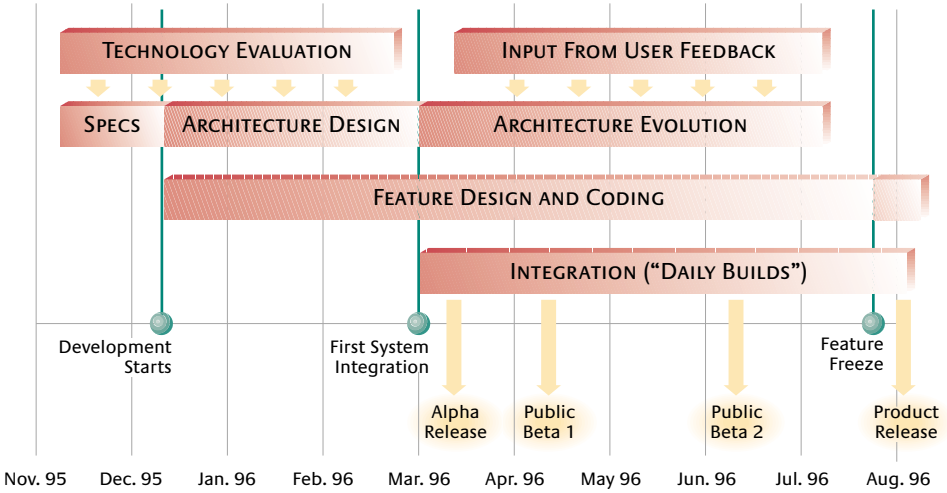
Consider Microsoft and its development of Internet Explorer. In the Internet's early years, small, nimble competitors such as Netscape and Yahoo! established leading positions — in part, through highly flexible development techniques.[7] In late 1995, many analysts thought Microsoft would be another incumbent that stumbled when faced with a disruptive innovation in its core business. Microsoft had been slow to recognize the potential of the Internet and was considered at least a generation behind Netscape in browser technology. Yet in the course of one project, Microsoft succeeded in making up the ground and introducing a product — Internet Explorer 3.0 — that many considered the equal of Netscape's offering. To a great extent, the achievement relied on the Explorer team's development process. (See "The Development of Internet Explorer 3.0.")

Internet Explorer 3.0 (IE3) was Microsoft's first browser release with a major internal-development component.[8] The project started on Nov. 1, 1995, with the white paper "How We Get 30% Market Share in One Year." A small team started putting together the initial specifications, which were released to Microsoft's development partners on Dec. 7. The project was designated a "companywide emergency." As one IE3 manager explained it, the designation meant that "if you were smart and had time on your hands, you should help out the IE3 team.

Given that we have a bunch of people here who are incredibly smart, we got a lot of great help. People realized this was a group that was going to determine what their stock was worth."

During December, detailed coding of the individual modules started. But the IE3 team was still making decisions about the overall product architecture — decisions that would not only affect the features in the final product but also the development process itself. A team member explained, "We had a large number of people who would have to work in parallel to meet the target ship date. We therefore had to develop an architecture where we could have separate component teams feed into the product. Not all of these teams were necessarily inside the company. The investment in architectural design was therefore critical. In fact, if someone asked what the most successful aspect of IE3 was, I would say it was the job we did in 'componentizing' the product."

**The Development of Internet Explorer 3.0**




The first integration of the new component modules into a working system occurred in the first week of March 1996. Although only about 30% of the final functionality was included in IE3 at that point, it was enough to get meaningful feedback on how the product worked. It also provided a base-line product, or alpha version, that could be handed to Microsoft's development partners. From that point on, the team instituted a process of "daily builds," which integrated new code into a complete product every day. Once new code was "checked in" (integrated into the master version), getting performance feedback through a series of automated tests typically took less than three hours. With the rapid feedback cycle, the team could add new functionality to the product, test the impact of each feature and make suitable adjustments to the design.

In mid-April, Microsoft distributed the first beta version of IE3 to the general public. That version included about 50% to 70% of the final functionality in the product. A second beta version followed in June and included 70% to 90% of IE's final functionality. The team used the beta versions (as well as the alpha version) to gather feedback on bugs and on possible new features. Customers had a chance to influence the design at a time that the development team had the flexibility to respond. A significant proportion of the design changes made after the first beta release resulted from direct customer feedback. Some of the changes introduced features that were not even present in the initial design specification.

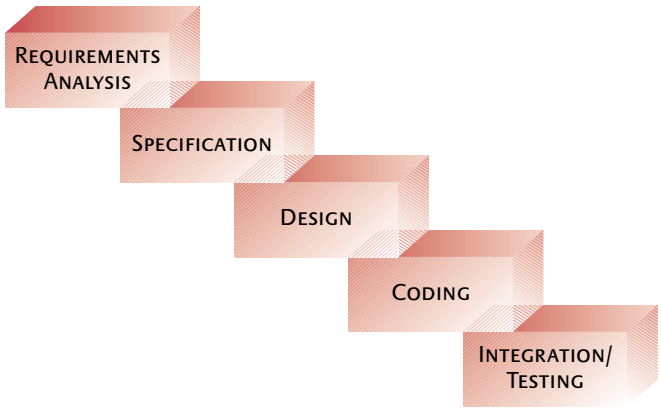
> **An Internet Explorer 3.0 team member declared, "If someone asked what the most successful aspect of IE3 was, I would say it was the job we did in componentizing the product."**

The cycle of new-feature development and daily integration continued frenetically through the final weeks of the project. As one program manager said, "We tried to freeze the external components of the design three weeks before we shipped. In the end, it wasn't frozen until a week before. There were just too many things going on that we had to respond to…but, critically, we had a process that allowed us to do it."

## Models of the Software-Development Process

The Explorer team's process, increasingly common in Internet-software development, differs from past software-engineering approaches. (See "The Evolution of the Evolutionary-Delivery Model," p. 78.) The waterfall model emerged 30 years ago from efforts to gain control over the management of large custom-software-development projects such as those for the U.S. military.[9] (See "The Waterfall Model of Software Development Is the Traditional Approach," p. 78.) The model features a highly structured, sequential process geared to maintaining a docu-

**The Waterfall Model of Software Development Is the Traditional Approach**




ment trail of the significant design decisions made during development. A project proceeds through the stages of requirements analysis, specification, design, coding, and integration and testing — with sign-off points at the end of each stage. In theory, a project does not move to the next stage until all activities associated with the previous one have been completed.

The waterfall model, which has been compared to ordering a mail-order suit based upon a five-page text specification, is best for environments in which user requirements (and the technologies required to meet those requirements) are well understood. Its application in more uncertain environments, such as Internet-software engineering, is problematic. Uncertain environments call for interactivity that lets customers evaluate the design before the specification has been cast in stone.

To achieve that objective, several alternative models use prototypes that are shown to customers early in the development process. Some companies employ a rapid-prototyping model that emphasizes the construction of an early prototype to help establish customer requirements.[10] Similarly, the spiral model moves through a series of prototype builds to help developers identify and reduce the major risks associated with a project.[11]

In both those models, however, the prototypes are not part of the design itself but merely representations that are thrown away after fulfilling their function. The bulk of the design work carried out thereafter is performed in a similar manner to the waterfall model.[12] In contrast, the process used to develop Microsoft's IE3 browser had at its heart the notion that a product can be developed in an iterative fashion. Critical parts of the functionality were delivered to customers early in the process; subsequent work added to the core design and responded to customers' feedback. Although the core functionality was continually improved, the early design that customers tested was an actual working version of the product.[13]

One development model with similarities to the IE3 process is the incremental, or staged-delivery, model.[14] In its basic form, it involves a system that is delivered to the customer in discrete chunks. However, it is unlike IE3's iterative process in that it assumes that the entire product design is specified in the early stages of development. Staged delivery is used only as a means of partitioning work so that some functionality can be delivered to customers early. By contrast, an iterative process is founded upon the belief that not everything can be known upfront — the staged delivery of the product actually helps determine the priorities for work to be done in subsequent stages.

The iterative process is best captured in the evolutionary-delivery model proposed by Tom Gilb.[15] In Gilb's model, a project is broken down into many microprojects, each of which is designed to deliver a subset of the functionality in the overall product. (See "The Evolutionary-Delivery Model of Software Development.") The microprojects give a team early feedback on how well the evolving design meets customer requirements. At the same time, they build in flexibility: The team can make changes in direction during development by altering the focus of subsequent microprojects. Furthermore, the number and length of the microprojects can be tailored to match the context of a project. In its most extreme form, each individual feature within a product could be developed in a separate microproject. To a large extent, the model mirrors the way IE3 was built.

## The Evolution of the Evolutionary-Delivery Model

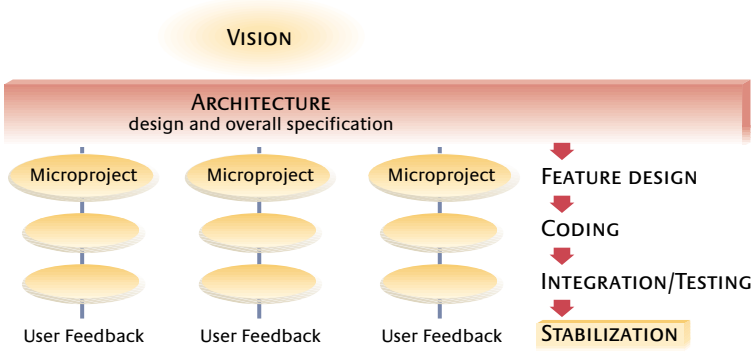
Companies that develop software are constantly improving the development models

- The Waterfall Model (a sequential process maintains a document trail)
- The Rapid-Prototyping Model (a disposable prototype helps establish customer preferences)
- The Spiral Model (a series of prototypes identifies major risks)
- The Incremental, or Staged-Delivery, Model (a system is delivered to customers in chunks)
- The Evolutionary-Delivery Model (iterative approach in which customers test an actual version of the software)

## Research on the Internet-Software Industry

Our study of projects in the Internet-software industry asked the question Does a more evolutionary development process result in better performance? The study was undertaken in stages. First, the researchers conducted face-to-face interviews with project managers in the industry to understand the types of practices being used. Next, they developed metrics to characterize the type of process adopted in each project. Finally, the metrics were incorporated into a survey that went to a sample of Internet-software companies identified through a review of industry journals. The final sample contained data on 29 projects from 17 companies.[16]

**The Evolutionary-Delivery Model of Software Development**




To assess the performance of projects in the industry, we examined two outcome measures — one related to the performance of the final product and the other to the productivity achieved in terms of resource consumption (resource productivity). To assess the former, the researchers asked a panel of 14 independent industry experts to rate the comparative quality of each product relative to other products that targeted similar customer needs at the time the product was launched.[17] Product quality was defined as a combination of reliability, technical performance (such as speed) and breadth of functionality. Experts' ratings were gathered using a two-round Delphi process (in which information from the first round is given to all experts to help them make their final assessment).[18] To assess the resource productivity of each project, the researchers calculated a measure of the lines of new code developed per person-day and adjusted for differing levels of product complexity.[19] Analysis of the data uncovered four practices critical to success.

**The most remarkable finding was that getting a low-functionality version of the product into customers' hands at the earliest opportunity improves quality dramatically.**

**Early Release of the Evolving Product Design to Customers** The most striking result to emerge from the research concerned the importance of getting a low-functionality version of the product into customers' hands at the earliest opportunity. (See "How the Product's Functionality at First Beta Affects Quality," p. 80.) The research provided data on the percentage of the final product functionality that was contained in the first beta version (the first working version distributed to external customers).[20] Plotting the functionality against the quality of the final product demonstrated that projects in which most of the functionality was developed and tested *prior* to releasing a beta version performed uniformly poorly. In contrast, the projects that performed best were those in which a low-functionality version of the product was distributed to customers at an early stage.

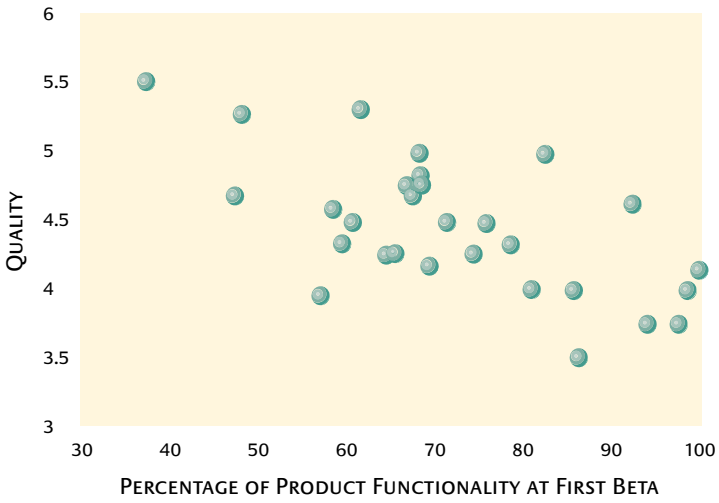
The differences in performance are dramatic. That one parameter explains more than one-third of the variation in product quality across the sample — a remarkable result, given that there are hundreds of variables that can influence the effectiveness of a development project, many of which are out of the project team's control.[21]

Consider the development of a simple Web browser. Its core functionality — the ability to input a Web-site address, have the software locate the server, receive data from the server and then display it on a monitor — could be developed relatively rapidly and delivered to customers. Although that early version might not possess features such as the ability to print a page or to e-mail a page to other users, it would still represent the essence of what a browser is supposed to do.

Of course, getting a low-functionality version to the customer early has profound implications for task partitioning. For example, let's say the aim of a project called BigBrain is to develop a new software application encompassing 10 major features. The traditional approach would involve dividing the team in such a way that all the features were worked on in parallel. Although progress would be made on each, the first opportunity to integrate a working version of the system would not occur until late in the project.

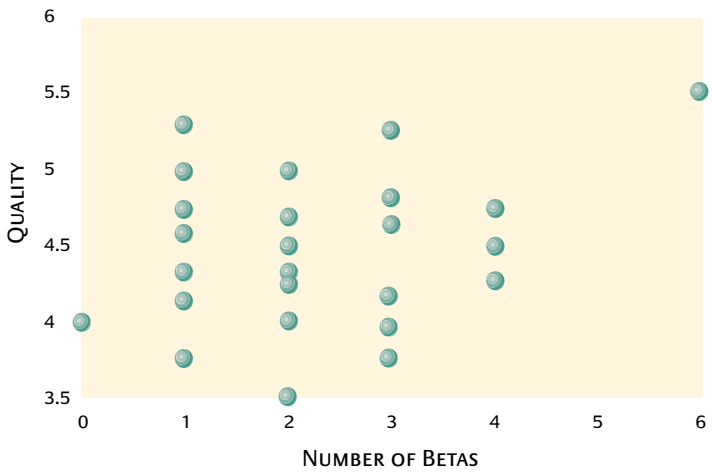
## How the Product's Functionality at First Beta Affects Quality

If customers test products early in development, when the products have low functionality, the final products are likely to have higher quality




## How the Number of Beta Tests Affects Quality

The number of beta tests an Internet-software-development group uses does not affect the quality of the end product




In an evolutionary process, however, the team might work first on only the three most important features — the essence of the system. Once those features were complete, the team would integrate them into a working version that could provide early feedback on how well the core modules interact. More important, the team would be able to distribute that early version to customers. As successive sets of features were completed and added to the product offering, their development would be guided by the customers' feedback.

The team might find that, of the seven remaining features planned for BigBrain, customers value only five (something customers may not have realized prior to testing a working version). In addition, customers might identify several features that had not previously been part of the design, giving designers the opportunity to make midcourse corrections — and thereby deliver a superior product.

By allowing the team to react to unforeseen circumstances, an evolutionary approach also reduces risk. Suppose that during the first part of project BigBrain, problems emerge in getting the core technical components to work together. With the evolutionary approach, the team can reschedule later-stage work — perhaps by eliminating one or more features of the original design. If development had proceeded in a more traditional fashion, feedback on such problems would not have been received until all the various component modules were integrated — much later in the process. The flexibility to react to new information would have been lost, and BigBrain would have shipped late.

Given the marked benefits of early beta testing, we considered whether the number of separate beta versions released to customers contributed to a product's performance. The Netscape Navigator 3.0 development team, for example, released six beta versions to external customers, each one following two to three weeks after the previous one.[22] The process of distributing an early release, gathering feedback, updating the design and redistributing the product to customers would seem an ideal way to ensure that the evolving functionality meshes with emerging customer needs. Surprisingly, however, the data showed no relationship between the performance of the final product and the number of beta releases. (See "How the Number of Beta Tests Affects Quality.")

Our interviews revealed that the benefits obtained from the evolving product's early release to customers depended not upon the number of releases but on the intensity with which companies worked with customers after the first release. In general, the number of releases was not a good proxy for how well a company chose its beta customers or how well they subsequently worked with those customers.

Indeed, although the project that produced the highest-quality product in the sample — Netscape Navigator 3.0 — released the

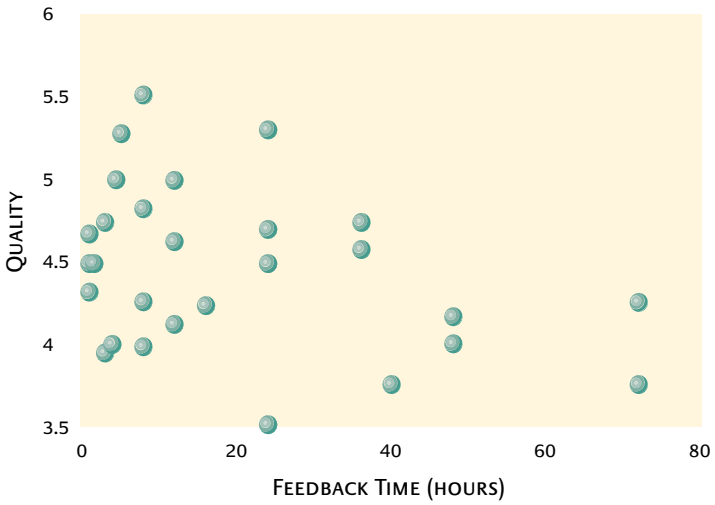
largest number of beta versions, a member of its development team noted that multiple versions can create version-control problems: "The majority of beta testers who give us feedback don't necessarily tell us which beta version they have been working with. So the problem is they might be pointing out a bug that has already been fixed or one that manifests itself in a different way in a later release of the product. The result is we can spend as much time tracking down the problem as we do fixing it."

**Daily Incorporation of New Software Code and Rapid Feedback on Design Changes** The need to respond to feedback generated through the release of early product versions to customers requires a process that allows teams to interpret new information quickly, then make appropriate design changes. In more than half the projects in the study, such changes were made through a daily build of the software code. In the same way that one checks books out of a library, developers working on the project would check out parts of the main code base to work on during the day. At the end of each day, they would take any newly modified code they had finished working on and check it back into the system. At check-in or overnight, a set of automated tests would run on the main code base to ensure that the new additions did not cause problems. At the start of the next day, the first task for each developer would be to fix any problems that had been found in his or her latest submissions.

Because daily builds have become an accepted approach to Internet-software development, they did not differentiate successful projects in the study. However, a measure of rapid feedback produced an intriguing result. (See "How Feedback Time Affects Quality.") We looked at final product quality and plotted it against the time it took to get feedback on the most comprehensive set of automated tests performed on the design. None of the projects with extremely long feedback times (more than 40 hours) had a quality level above the mean. The conclusion, supported by interviewees' comments, is that rapid feedback on new design choices is a necessary component of an evolutionary process. However, rapid feedback alone is not sufficient to guarantee that evolutionary software development will result in success. Indeed, projects with short feedback times were just as likely to perform poorly as to perform well.[23]

**How Feedback Time Affects Quality**

Rapid feedback on changes made to software facilitates better product performance




**A Team With Broad-Based Experience of Shipping Multiple Projects** One might be forgiven for thinking that the value of experience is limited in a revolutionary environment such as the Internet-software industry. Much academic research has pointed out that in dynamic environments, experience may cause trouble, given that the knowledge of specific technologies and design solutions atrophy fast.[24] Indeed, the development ranks of many leading Internet-software companies often are filled with programmers barely out of college. Yet the view that a less experienced team is somehow better at developing products in such environments defies common sense — after all, a team with no experience at all would not know where to start. Even in a development process with the capacity to run thousands of design experiments, there is still a need to decide *which* experiments to run — and then to interpret the results. The question that must be asked therefore is What *types* of experience have value in revolutionary environments?

**A youthful development team is not necessarily one that lacks relevant experience.**

To answer that question, we studied two different measures of experience. The first was associated with the more traditional view of experience — the average tenure of the development team in years. The second measure reflected a different form of experience — namely, the number of project generations a team member had completed (generational experience).[25]

Our thinking was that the experience of completing many different projects would give developers a more abstract

approach — one that evolves over time as lessons are learned from successive projects. As a result, they would be better equipped to adapt to novel contexts and applications. In addition, the completion of each project would help developers see how their work fit into the system. The more projects completed, the greater their knowledge of how to design effectively at the module level while keeping the system level in view.

The results showed that the traditional measure of experience had no association with either product quality or resource productivity. The measure of generational experience had no association with product quality either, but it turned out to be a powerful predictor of resource productivity.[26] This suggests that the value of completing multiple projects in an evolutionary-development environment does not derive from an ability to predict specific customer requirements. Rather, such experience — by providing knowledge that helps developers analyze and respond to the data during development — allows greater efficiency in ongoing design activities.

The findings provide some insight into why a youthful development team is not necessarily one that lacks relevant experience. Given that many software-development projects have short lead times, it is possible for a developer to complete quickly a large number of projects, thereby gaining substantial generational experience. That experience benefits future projects by helping the developer frame and direct an experimentation strategy that can resolve design problems quickly, even when the problems faced are novel.

**Major Investments in the Design of the Product Architecture** In most development projects, the main design criterion for the product architecture is that it provide the highest possible level of performance. Often, the way that occurs is through an architecture that tightly links the various component modules. In an evolutionary process, however, there is another important criterion — flexibility in the development process. Designing the architecture so that a version of the product can be assembled at an early stage and distributed to customers requires explicit architectural choices. Building in the ability to accept additional functionality during late project stages adds further demands.

**A valuable avenue for identifying beta partners is through a customer-support database and through asking employees which customers experience the strangest problems.**

The key to an evolutionary process is to develop an architecture that is both modular and scaleable.[27] A more modular system is better at accommodating changes to the design of individual modules without requiring corresponding changes to other modules in the system. The loosely coupled nature of a modular system buffers the effect of changes. It is therefore suited to uncertain environments — at least, to the degree that the design isolates uncertainties within modules and does not allow them to affect the interfaces between modules. A more scaleable system allows initially unanticipated functions and features to be added at a late stage without disrupting the existing design. That requires a solid underlying infrastructure, such as that of the Linux operating system, one of the best examples of a modular and scaleable architecture. (See "A Modular and Scaleable Architecture: The Linux Operating System.")

In lieu of being able to examine the details of each product's architecture for our study, we focused on the relative investments in architectural design that companies made.[28] Our assumption was that those investments reflected the degree to which companies were trying to resolve potential conflicts between a highly optimized architecture (one that is tightly coupled) and an architecture that facilitates process flexibility (one that is both modular and scaleable). Our analysis confirmed that a high level of investment in architectural design did indeed have a strong association with higher-quality products.[29]

## Putting It All Together

Although the study demonstrated that early customer involvement in an evolutionary process is vital, companies must take care to select suitable beta partners. We learned from the fieldwork that a valuable avenue for identifying beta partners is through exploring a company's customer-support database to identify customers who stretch the performance envelope. Customers who initiate numerous calls for support are good candidates for beta programs; however, it is the *nature* of those calls that is critical. The most effective beta groups include distinct customers for each performance dimension (say, reliability, speed or functionality) rather than customers who make demands on many fronts. Asking support employees which customers experience the strangest problems is one way of identifying those who are using the product in novel ways and who therefore might provide useful insights on performance.

With regard to the number of beta customers involved, we noticed that some companies emphasized a broad release of early versions to the entire customer base, whereas others employed a narrower distribution to a select group. The former strategy

## A Modular and Scaleable Architecture: The Linux Operating System

The initial version of the Linux kernel, the core of the Linux system, was developed in 1991 as part of an open-source project to develop a freely available Unix-like operating system. (In open-source projects, the underlying code that makes the software work is distributed to users so they can improve upon or customize it.) At the time, the kernel comprised only 10,000 lines of code and ran on only one hardware platform — the Intel 386. However, as developers around the world began using the system and contributing to the project, its functionality expanded dramatically. By 1998, the kernel had grown to more than 1.5 million lines of code and was used on hardware platforms from supercomputers to robotic dogs. (See "Evolution of the Linux Kernel.") Yet estimates of the amount of code added by the system's originator, Linus Torvalds, were typically less than 5%.

**Evolution of the Linux Kernel**

| Year | Version | Lines of Code | Users |
|---|---|---|---|
| 1991 | 0.01 | 10,000 | 1 |
| 1992 | 0.96 | 40,000 | 1,000 |
| 1993 | 0.99 | 100,000 | 20,000 |
| 1994 | Linux 1.0 | 170,000 | 100,000 |
| 1995 | Linux 1.2 | 250,000 | 500,000 |
| 1996 | Linux 2.0 | 400,000 | 1,500,000 |
| 1997 | Linux 2.1 | 800,000 | 3,500,000 |
| 1998 | Linux 2.1.110 | 1,500,000 | 7,500,000 |

Source: Forbes, Aug. 10, 1998.

When Torvalds released the original version of the code, he could not have predicted the functionality that would be added in subsequent years. So he based the design upon a modular architecture even though the inner core was monolithic. Developers around the world would be able to contribute to Linux without having to worry about the effect their code would have on other modules. Torvalds also made Linux scaleable — able to accept new functionality in a way that minimized changes to the existing core. Although the Linux architecture owes much to the long heritage of Unix, it is also a reflection of the true genius of its author.

seemed most useful for product segments in which the software was meant to operate on a variety of different hardware platforms and alongside other software products. In such situations, mass distribution helped identify bugs related to the interactions among systems and products. In contrast, enterprises with sophisticated products that placed greater demands upon users preferred to work with a smaller group, given the extra support that such users required.

The benefits of an evolutionary approach to software development have been evangelized in the software-engineering literature for many years. However, the precise form of an evolutionary model and the empirical validation of its supposed advantages have eluded researchers. The model has now been proved successful in the Internet-software industry. When combined with the insights gained in fieldwork, our research suggests a clear agenda for managers: Get a low-functionality version of the product into customers' hands at the earliest possible stage and thereafter adopt an iterative approach to adding functionality. The results also underscore the importance of having a development team with experience on multiple projects and creating a product architecture that facilitates flexibility.[30]

The usefulness of the evolutionary model extends beyond developing software in environments with rapidly changing markets and technologies. By dividing tasks into microprojects, a company can tailor the process to reflect any particular context. Uncertainty in the Internet-software industry dictates short microprojects — down to the level of individual features. Traditional market research has limited value here, so companies need an early working version to gain feedback on the product concept.[31] In more-mature environments, however, companies can specify more of the product design upfront, use longer microprojects and develop greater functionality before needing feedback. In a world where customer needs and the underlying technologies in a product are known with certainty, only one large microproject is necessary, and the waterfall model suffices. An evolutionary-delivery model represents a transcendent process for managing the development of *all* types of software, with the details tailored to reflect each project's unique challenges.

### ADDITIONAL RESOURCES

Readers interested in the general topic of managing product development are directed to a popular textbook, "Revolutionizing Product Development," by Steven Wheelwright and Kim Clark, published in 1992 by Free Press. The most practical publication specifically on software development may be the 1996 Microsoft Press book "Rapid Development," by Steve McConnell.

A deeper discussion of the open-source approach can be found at www.opensource.org/. To read more about Linux and one of the companies involved in its distribution, see the Harvard Business School case "Red Hat and the Linux Revolution," by Alan MacCormack, no. 9-600-009.

For a discussion of Microsoft's approach to developing software, see "Microsoft Secrets," by Michael Cusumano and Richard Selby, a 1995 Free Press book. Harvard Business School's multimedia case "Microsoft Office 2000," by Alan MacCormack, illustrates that approach in detail (case no. 9-600-023), and the accompanying CD-ROM contains interviews with team members and a demonstration of Microsoft's Web-based project-management system.

A new model of software development with similarities to the evolutionary model is "extreme programming." Details can be found at www.ExtremeProgramming.org/. The Software Engineering Institute at Carnegie-Mellon University is a useful source of research on software-engineering management. See www.sei.cmu.edu/.

## REFERENCES


**1.** "Forecasting a Robust Future," www.bsa.org/statistics/index.html?/statistics/global_economic_studies_c.html.

**2.** Measured in terms of value added. Ibid.

**3.** Ibid.

**4.** "A Survey of the Software Industry," The Economist, May 25, 1996, 14.

**5.** See, for example: M.A. Cusumano and R. Selby, "Microsoft Secrets" (New York: Free Press, 1995); and F.P. Brooks, "The Mythical Man-Month" (Reading, Massachusetts: Addison-Wesley, 1995).

**6.** A. MacCormack, R. Verganti and M. Iansiti, "Developing Products on Internet Time: The Anatomy of a Flexible Development Process," Management Science 47, no. 1 (January 2001).

**7.** M. Iansiti and A. MacCormack, "Developing Products on Internet Time," Harvard Business Review 75 (September-October 1997): 108-117.

**8.** The first two versions of Internet Explorer relied extensively on licensed technology.

**9.** W.W. Royce, "Managing the Development of Large Software Systems: Concepts and Techniques" (Procedures of WESCON [Western Electric Show and Convention], Los Angeles, August 1970).

**10.** J.L. Connell and L. Shafer, "Structured Rapid Prototyping: An Evolutionary Approach to Software Development" (Englewood Cliffs, New Jersey: Yourdon Press, 1989).

**11.** B. Boehm, "A Spiral Model of Software Development and Enhancement," IEEE Computer 21 (May 1988): 61-72.

**12.** For example, in Boehm's spiral model, the outer layer of the spiral contains the activities of detailed design, coding, unit testing, integration testing, acceptance testing and implementation. Those activities are carried out sequentially.

**13.** That does not preclude the fact that "throwaway" prototypes are used in such a process. Indeed, they are likely to be extremely important in establishing a direction for the initial design work.

**14.** See, for example: C. Wong, "A Successful Software Development," IEEE Transactions on Software Engineering (November 1984): 714-727.

**15.** T. Gilb, "Principles of Software Engineering Management" (Reading, Massachusetts: Addison-Wesley, 1988), 84-114.

**16.** The survey was distributed to 39 firms, of which 17 responded with data on completed projects. The resulting sample of products is quite diverse and includes products and services targeted at both commercial and consumer users.

**17.** Quality was assessed on a seven-point scale, with level four indicating the product was at parity with competitive offerings.

**18.** H.A. Linstone and M.Turoff, eds., "The Delphi Method: Techniques and Applications" (Reading, Massachusetts: Addison-Wesley, 1975).

**19.** Projects in our sample differed significantly with regard to the number of lines of code developed. We therefore normalized the resources consumed in each project to reflect the development of an application of standard size. We adjusted the resulting measure for scale effects (larger projects were found to consume relatively fewer resources) and complexity effects (projects to develop Web-based services were found to consume relatively fewer resources, because of the specifics of the programming language used).

**20.** A beta version, as defined, is not a throwaway prototype. It is a working version of the system. The measure of the percentage of product functionality contained in the first beta was adjusted for scale effects.

**21.** We also examined the relationship an early beta release has with resource productivity. One might have imagined that in an evolutionary process there is a penalty to pay in terms of productivity, given the possibility that some early design work will be thrown away as customer requirements become clearer. However, our results showed no association between an early release to customers and lower productivity. The benefits from an early release appear to overcome the potential drawbacks of multiple iterations.

**22.** M. Iansiti and A. MacCormack, "Developing Products on Internet Time," Harvard Business Review 75 (September-October 1997): 108-117.

**23.** As a result, the correlation between feedback time and product quality is not statistically significant.

**24.** See, for example, R. Katz and T.J. Allen, "Investigating the Not-Invented-Here (NIH) Syndrome: A Look at the Performance, Tenure and Communication Patterns of 50 R&D Project Groups," in "Readings in the Management of Innovation," eds. M. Tushman and W. Moore (New York: HarperBusiness, 1982), 293-309.

**25.** We used the term "generations" to distinguish between major "platform" projects (that is, those in which major changes were made to the previous version of a product) and minor derivative/incremental projects.

**26.** The measure of generational experience we used in our analysis was the percentage of the development team that had previously completed more than two generations of software projects. Note that the variation in generational experience explains more than 24% of the variation in resource productivity.

**27.** Note that there is a relationship between those two characteristics. Namely, a scaleable architecture is likely to be modular. A modular architecture, however, is not necessarily scaleable.

**28.** The measure we used, adjusted to control for scale effects, was a ratio of the resources dedicated to architectural design relative to the resources dedicated to development and testing.

**29.** Those investments explain more than 15% of the variation in product quality. We found no significant association between them and differences in resource productivity.

**30.** In our sample, measures of the parameters in combination explain almost half the variation in product quality and a quarter of the variation in resource productivity.

**31.** For example, consider attempting back in early 1996 to conduct market research into the features that a browser should contain. Most people would have had no clue what a browser was meant to do. Hence traditional market research techniques (focus groups, surveys and the like) would have had less value.


---


**Reprint** 4226