2

Budget and Schedules Aren't Ideal

If the only problems a Web application designer had to surmount were technical, solutions would be simple. Men have walked on the moon, a computer has beaten a Grand Master at chess, and entire Thanksgiving dinners can be made with soy products. There's never been a technical problem that couldn't be overcome by extraordinary amounts of technical effort, but the real difficulty lies in applying technical solutions to real-world problems without expending the same effort over and over.

The problem, of course, is that the resources we have available to expend on technology are limited. In fact, they're limited even further by the primary goal of technology, which is to make our lives easier and more productive as well as to make new things possible. This mantra means that expending a decade's worth of effort to put a man on the moon was a wonderful thing, but expending an additional decade's effort for each person that wants to go to the moon would be out of the question.

On the Web, the problems are similar. If all users had broadband connections, and if all servers were supercomputer clusters with OC-12 lines, and if all webmasters had infinite amounts of time and money to throw at a problem, there would be no difficulty in providing everyone with a rich, rewarding, and productive Web experience.

Unfortunately, the reality is that computers are obsolete as soon as they're out of the box, customers are demanding more features that are easier to use and that consume less bandwidth, and your network is starting to get old and creaky. To top it

off, the boss is asking for a whole slew of new buzzword-compliant technologies for every Web application within the company. Oh, and it all has to be done by yesterday, thanks. It's enough to send a sane Webmaster around the bend.

The Web environment also creates problems that are even less responsive to technical solutions. Web applications are likely to involve custom programming that brings together disparate technologies, departments, and people to address an issue that they all have in common. Of course, this is the same problem that hundreds of years of business planning and organizational theory have been unable to solve. So, it seems logical to add an organizational solution to all the technological requirements. As a result, the Webmaster is also taskmaster, reporter, presenter, and arbitrator. In addition, all this must be done in less time, for less money, and with fewer requirements on the rest of the staff because they're busy doing important things.

Slow Decisions Versus Fast-Approaching Deadlines

Corporations are slow about nearly everything. It's a fact of life in the business world, and there's very little that can be done to change the decision-making process of a 7,000-employee business that stands to lose billions of dollars if it makes the wrong choice. On the other hand, every individual in a corporation demands that everything under his or her control be available as soon as possible to try and combat the overall slowness of the organization. If the other departments are slow and unresponsive, the reasoning goes, at least one group's results can be made available yesterday.

A bureaucracy is an anathema to the Web application design process, which requires quick thinking, flexibility, and time to produce a Web experience that is on par with the user interfaces that site visitors have come to expect. The price of failure is high on the Web; people go elsewhere if their needs aren't being met. If it takes six months before a new service is made available to customers, that might be the six months necessary for a fast-moving competitor to start, get funded, develop a revolutionary technology, and provide a service that outstrips the one offered by the slower company. The result is obsolescence and downsizing. Welcome to the Web.

The Weight of Progress

On the Web, new technologies are made, used, and overridden within the span of a few months. As a result, a four-to-six month span of time is called a *Web year*, with the implied notion that new products, services, and companies come out every "year."

With all this churn, it's difficult to stick to the basic goals of a Web application project. If the project was originally supposed to provide a service for home buyers looking for a loan over the Web, for instance, a whole host of new technologies and services might become available by the time the project is underway. New partnerships with realtors, new links to lender networks and their technologies, and new vendor contracts can all be made while the initial goals of the project are being implemented.

Management could also demand new features from a Web application based on interest from customers or directors, regardless of whether the new features are easy to implement.

Technology is usually outdistanced by the hype surrounding it in the press. As this book is written, there are many technologies, standards, and industry movements that are popular in the press but completely unimplemented. The Universal Description, Discovery, and Integration (UDDI) specification, for instance, is a standard being touted by IBM, Microsoft, Hewlett-Packard, and others as the answer to directory listing of automated Web services. But, as of January 2001, there were few companies listed in the UDDI databases and only one Web service. Meanwhile, companies around the world are scrambling to build UDDI compliance into their products because they're being told that it's the next big thing. If it fails instead, which it has 50-50 chances of doing, those companies must write off all UDDI-related development and start fresh with the next standard touted by the big players.

The result of all this progress is a Web environment that uses a lot of "pre-release" software. The Web server implementation community has become a nation of early adopters and technology implementers who have to take whatever technology is available as long as it performs even part of its intended job. Gone are the days when software was expected to work right out of the box; the need for an army of consultants and custom programmers now is assumed when implementing a Web application project. In fact, expecting instant usability from these programs has become so unusual that programmers and implementers are surprised when a program works as advertised without any unpleasant side effects.

Evaluating New Technology

The evaluation process is rarely a smooth one; it tends to unearth more issues than it solves. Evaluating new technology or technology upgrades would be simple, if only one person were doing all the evaluating, deciding, and implementing. Unfortunately, the managers with the purchasing power usually make the decision, and the engineering staff must implement it. The task of evaluating a solution tends to be assigned to a mix of the two groups with varying degrees of authority given to each group. In reality, the decision-making process is influenced as much by vendors, consultants, media perceptions, and politics as it is by feasibility.

Decision-making isn't made easier by vendors and consultants. No vendor wants to explain the aspects of its technology that don't work, and very few even acknowledge that there are situations for which their products aren't suited. No vendor mentions the platforms they don't support, the aspects of the program that are still in beta, or the reasons why other products are faster or more reliable. Consultants, at least those sponsored by a particular vendor, aren't much different. Their solutions always are biased toward their sponsors' products, no matter how bad the fit might be; their solution is a hammer—so every problem must be a nail.

Independent consultants can provide an opinion free of corporate bias, but their pronouncements might be geared more toward the perceptions of the client than toward technical reality. They might recommend a single suite of products that offers partial solutions to each client need, rather than explaining the individual roles of disparate technologies that would solve the problem more effectively.

Even reviews and articles in the mainstream media can be misleading when evaluating a new technology. Trade journals and computer magazines show a bias toward products and technologies that are new and exciting. These new technologies provide an endless supply of new specifications and marketing descriptions to use as source material. The source material eventually becomes articles that describe the new technology to readers who have never encountered it.

On the other hand, old technology—especially stable, ubiquitous technology—isn't news because it provides limited draw to introductory or speculatory articles about it. Thus, "push" technology was all the rage in the press while the Apache Web server made quiet inroads into thousands of sites, and every magazine gushed about VRML (the ill-fated Virtual Reality Modeling Language) while database-backed Web applications generated real income. The continued disparity between the popular conception of hot new technologies and the technical reality causes real conflict when decision-makers are doing the reading and engineers are doing the implementing.

Even after finding a suitable technology, it's possible that political decisions will make it unavailable for a particular project. This technological standoff often occurs within project groups that are formed by merging the talents of several existing groups, for instance when companies merge or are taken over. Each group might have its own solution set implemented using a different technology, but consolidation efforts invariably require one group to abandon their existing solutions in favor of "standardized" technology, even when the solutions would not be technically incompatible. This forces entire groups of developers to use technology that is, at best, unfamiliar to them, and, at worst, unsatisfactory to the their desired ends.

Sometimes, engineers have to use a technology unofficially because of the politics surrounding it. For years, UNIX servers had to "go underground" because the prevailing view was that Windows NT would overwhelm the server market and provide everything the UNIX servers did and more. Magazines and trade journals declared the death of UNIX weekly, and managers spoke of the inevitability of their switch to Microsoft products. Developers who saw UNIX as the best solution for their particular needs were forced to implement servers quietly while giving lip service to their "transition" to Windows. Fortunately, the exponential growth of Linux as a server environment gave a boost to the visibility of UNIX in the media, and stories about the death of UNIX tailed off considerably. Interestingly enough, after this happened, corporations started noticing that many of the systems they thought had been implemented using Windows were actually Linux systems masquerading as such.

Dealing with Unexpected Decisions

All the plans and decisions made when designing Web applications can be crushed with a few simple phrases like these: "We're standardizing on Oracle," "We're going to use Windows from now on," and "Everything has to be made available to WAP phones." Snap decisions about technology are always a possibility in an environment in which new insights from partners, consultants, and investors can change the technology direction of a project overnight.

Changes in vendors can be the most damaging to a project. The decision to switch database vendors, for instance, might require all database-backed Web applications be rewritten to support differences in query syntax between the old and new databases, even though the vendor might insist that the database is "fully standards compliant." What this really means is that the old and new databases support the same core standard, but that they provide their own "enhancements" to it. Although some sudden decisions can be averted by knowing the true nature of the changes involved, many times these decisions are made without developers' knowledge. The decision to switch vendors can also be affected by company mergers and purchases; when this happens, the company usually decides to use the newly-acquired product, regardless of whether it's compatible with what's currently being used.

Decisions that change the platform requirements of a project are easier to deal with when developing Web applications. Web technologies were designed from the start to be as standards-compliant as possible, and that history has fostered the need for cross-platform tools as the basis of Web technologies. This need also has been helped by the UNIX-centric nature of Web application environments; UNIX was designed to be portable, with little dependence on hardware or specific implementations. As a result, core Web technologies like the Web server, Perl, Java, and the Web browser itself are available on almost every server platform, and derivative standards like TCP/IP, HTTP, XML, and HTML have implementations across many platforms and environments. Even some proprietary systems, such as Oracle, have few differences between versions for different platforms. Thus, it's possible to change server platforms without impacting the design of a dependent Web application. The few exceptions include Microsoft server technologies, such as ASP and COM, which tend to tie themselves to Microsoft platforms as much as possible. Other exceptions include MacOS platforms before MacOS X. These platforms had limited support for common Web application tools. The limited support problem is addressed by MacOS X and future MacOS systems, all of which are based on BSD UNIX and all of which integrate standard services such as the Apache Web server.

Adding an additional technology to a specification can be easy or difficult to implement, depending on the scope of the technology involved and the difference between the technology's requirements and the requirements of the rest of a project. For instance, adding support for WAP phones, as discussed in Chapter 17, "Publishing XML for Wireless Devices," can be very simple from the standpoint of the Web server; however, it requires a duplication of effort from the standpoint of Web content

production. This difference might not be factored into the decision-making process, however, and consultants or articles might gloss over the potential difficulties when describing the benefits. In the case of WAP phones, articles might emphasize the idea of "the Web on a phone" and the number of additional clients this provides. However, the articles might not emphasize the fact that most Web content has to be specially repurposed for the WML browsers on these phones.

The best protection against change is selecting technology that is flexible and that interacts with a multitude of potential systems—not just the ones originally present in a specification. Even the most flexible framework is not compatible with every new technology and delivery mechanism, but many trends in Web application design can be anticipated well in advance of their inclusion in project specifications. So, it's reasonable to design the current generation of Web applications with the next generation of Web technologies in mind. This idea is explored in Section III of this book, "Solutions for the Future."

Small Budgets Versus Grand Ambitions

Decisions are usually made more difficult by the lack of money for technology purchases and implementation. Sometimes, the best solution to a technical problem is prohibitively expensive, and partial solutions have to be implemented instead. For Web applications, this often happens when evaluating server hardware and network infrastructure. Vendors would love to sell every Web application designer a colocated, load-balanced server cluster with hardware fail-over and near-infinite bandwidth; however, in reality, many Web applications have to provide the same performance on a single low-budget server that operates through a less-robust connection to the Internet.

Of course, the lack of money doesn't stop decision-makers from dreaming up the largest possible set of features for a Web application to support. A system with more features is always more impressive than a system with fewer features. In fact, new features are more likely to be added than old ones are to be taken away. When this happens, you must keep track of the features that are most essential to a project. It's also helpful to set proper expectations within the time and money allocated to a project. You also should budget for the inevitable flood of new features that get proposed when initial versions of a Web application come available. This flood happens because users always find uses for a Web application that require new features to be implemented.

Making Room for Essentials

Even the most poorly defined project has a core set of goals that have to be met at a basic level to satisfy everyone involved. No matter how many wish lists and feature requests are added to a project's scope, these main goals are the things without which the project can't be considered a success.

Hardware and software requirements are usually the first set of essentials to be defined, but there's more flexibility in implementing these solutions than would usually be acknowledged. For instance, a decision to use the Solaris operating system on Sun server hardware might seem to imply a fixed cost, as determined by Sun's current pricing model for performance that a Web application demands. It's often possible, however, to gain additional performance by the efficient use of server software. This software can range from clustering software on multiple systems to optimized software for older systems. In addition, it can enable less-expensive choices to be made while still meeting the core requirement. As another example, the seemingly-fixed cost of doubling the amount of RAM in a server can be reduced by reducing the memory requirements of the Web application software that the server runs. The former incurs a monetary cost, but the latter uses development time to achieve the same result. If a project has more development time than money to spare, the decision becomes clear. (Perl modules, open source software, and the techniques in this book are examples of using the development time of others as well. By implementing solutions other people have developed to reduce hardware requirements, it's possible to save money without a penalty.)

When determining the features of a Web application that should be implemented first, it's wise to consider how these features could share common technology components. This consideration enables a solution to be implemented and tested with one feature and then adapted to related features as they are implemented. Generally, a large group of features can be found that implement a common technology, which can then be included in the list of essentials with highest priority. Even though the common technology might take up a large chunk of the budget, after that core technology is implemented, new features of that type can be implemented more quickly and cheaply. As an example, consider that implementing a publishing-centric site around XML technologies might cause additional work and require more technology purchases to assure high performance in XML parsing and translations. The decision would be warranted if new features such as syndication or wireless device support could be added for a fraction of the cost in a fraction of the time. Of course, not all features are easily grouped into classes, and it's sometimes better to implement a quick solution rather than attempt to fit radically different features into a common framework. It's perfectly acceptable to implement a feature multiple times in different ways instead of spending the same amount of time finding a common solution to all possible implementations.

Setting Expectations

When developing a project, it's important to make sure that decision-makers and end users have an accurate view of the work that will actually be done and the problems that will actually be solved. Setting the proper expectations can make the difference between a project that is considered successful and one that is sent back to the drawing board.

The idea of setting expectations is one that independent contractors learn very early in their careers. If a client asks for the moon on a platter originally but is convinced by the contractor to accept a more reasonable solution within the scope of the project, it's often the case that the client reaffirms his or her need for the moon when the product is delivered. It's as if the client doesn't remember the "reasonable solution" discussion. Without a written record of the expectations that were set originally, many contractors would not get paid until the client exhausted every possible feature desired in a product.

For developers working within an organization, it's possible to get the same effect by making the accepted scope of a project explicit at every opportunity. When developing an application that will be accessible by Web browsers and Palm devices but not WAP-enabled cell phones, for example, you might need to state that up front and repeat it frequently to remind decision-makers and managers that the project has a limited scope. If the proper expectation is set originally, later changes to the scope of the project can be seen as additions to the project. These addition can be decided upon and budgeted for separately. In the WAP example, if managers see an increased need for WAP devices as well, the decision to add support for WAP devices can be evaluated and the appropriate budget can be allocated. More often than not, separating out features in this way can keep superfluous and expensive features from sneaking into a specification after the project is already underway.

Living Up to the Demo

Sometimes, success can inflict its own punishment, even under the best development circumstances. Interest in a project can seem impossible to generate initially, and feedback from early users and testers is great for usability testing. After the project is successful, though, it's possible to get inundated with requests for product fixes and new features.

A demonstration of product capabilities can offer a concrete example of how a Web application can be used. Until users get a chance to see the program in action, it might be difficult for them to visualize the way in which it really works. Seeing the demo gels the application in their minds and gives them a basis by which to evaluate their plans for using it. The demo also gives users a chance to express preferences that they wouldn't have been able to articulate without specific examples in front of them.

Unfortunately, a successful demo also can give rise to usage ideas that weren't anticipated when choosing the initial scope of the project. Viewing a demo can give users more insight into the workings of the application, but it also gives the impression that the application is complete and looking for new ideas. Instead of giving feedback on the technologies demonstrated, users are likely to comment on a dozen other tangential solutions they'd like to see from that Web application. If each tangential solution is added to the project's list of required deliverables, the project can quickly drown.

Initial demonstrations of a developing Web application should be seen as a part of the decision-making process as well as a time to do informal usability testing. User feedback is important, but only when it fits in with the established goals for the project. Does solving the user's issue require changing the focus of the Web application, and would the change require delaying the application's release? If so, it might be better to include the requested feature on a list of revisions to carry out after the initial release. To avoid growing a long list of these, which might detract from the success of the application as users wait for the next release, it usually helps to keep the scope of the application flexible enough to allow at least some features to be implemented quickly, if requested.

Some Help is No Help at All

It starts innocuously, but it ends up being the downfall of your project. You ask for a graphic designer, and the boss brings in his 11-year-old daughter who's "a whiz at these things, really." You ask for a new Web server, and the boss brings out a 10-year-old minicomputer from the depths of the server closet. You ask for usability studies, and the boss shows the Web application demo to his buddies at competing companies and gives them your phone number. You ask for timely content, and the boss gives you old data in an archaic format stored on a backup tape that might or might not be readable. Every time you ask for help, in fact, you're given another technological challenge instead of solutions for the current ones.

Some help is truly no help at all. A Web application designer looking to provide a cutting-edge interface using the latest technologies—on time and under budget—can be hampered as much as helped by old data, old systems, staff with indeterminate Web experience, and unwanted feedback from an audience that isn't likely to use the site regularly. In many cases, though, this is the only help that's available. The old systems are the only hardware budgeted for the project, there's no money to hire staff with actual Web knowledge, and a usability testing firm would be entirely too expensive. So, the best has to be made from the materials at hand.

Working with Legacy Systems

Legacy data is a term that's tossed around a lot in the industry, but it basically means "data from any software that isn't directly supported by this software." By lumping all external software into a category with data tapes and punch cards, software vendors can escape the responsibility of interacting with many of the systems to which their customers would ordinarily like to connect.

A Web application designer doesn't have the luxury of dismissing legacy data, even when the tools used to develop Web applications do. Never mind that there isn't an Enterprise Java Bean built to read records from old FoxPro databases; if that's where the data is, it needs to get to the Web somehow. If a company's business logic is contained in an IBM mainframe accessed through CICS, the Web application had better speak the language and operate on the resulting data in a meaningful way.

These data connection needs are where the choice of a Web application environment is most important. Does the environment support adding additional sources of data? How easy is it to do so? Better yet, is the existing base of support for data sources broad enough that adding support for legacy sources is both rare and trivial? These concerns are important when you are evaluating Web environments (servers, programming languages, and middleware suites). No matter how easy an environment makes the core 90 percent of Web application design, it's not a good choice if the other 10 percent is difficult enough to erase any productivity gains.

Delegating Web Development

Web application technology aside, it's a good idea to let your Web application benefit from the work of others. Graphic artists, system administrators, database administrators, user interface designers, copy editors, and other potential contributors should all be given the easiest route possible to offering input and making improvements to a Web application.

Delegating aspects of Web application development requires planning and forethought. If it's difficult to make graphic and layout changes to the pages of an application, for instance, those changes are unlikely to be made in a timely manner. Designers and usability testers will be discouraged from making modifications freely, and the application interface suffers as a result. See Chapter 3, "Site Design Versus Application Design," for further discussion of this subject.

Involving the Audience

Any Web application, be it a public Web site or a private intranet application, has a built-in source of data that's rarely acknowledged when building the application—users. Since the scope of the site assures that the primary audience has at least one thing in common, it's important to allow users to modify application structure and contents to the full extent possible. The more work users do in shaping a Web application, the closer it's likely to be to the application they'd prefer. Besides, any modification or addition that users make to a Web application without developer support is an additional feature added to the application for free.

On the other hand, giving users that level of control over a Web application can require as much or more development work as creating the application in the first place. User administration forms and other site meta-interaction pages are designed the same as any other user input; however, there are additional concerns because of the more fundamental changes users are making to the site. Allowing visitors to a news site to add headlines to the front page, for instance, also brings the risk that any user input might modify the page in a way that is aesthetically unpleasant at best and disastrous at worst. You can mitigate these concerns by defining explicitly what a user is allowed to add or change. You want the controls as tight as the content allows, but you should remember that accounting for every unwanted circumstance can be time-consuming.

Some users can be a boon to a Web application. In the news site example just discussed, no mention was made about the way headlines get on the home page if site viewers aren't submitting them. In many cases, site personnel or other privileged Web application users need to use the same interfaces that unprivileged site visitors would use. So, the choice is not between offering and not offering user interfaces, but between offering interfaces only to a few users or putting the time in to make them available to all.

Summary

Not all problems are technological, and not all technology solutions are affordable or feasible in the time allotted for a project. New technologies are vital to a robust Web project, but any new technology must be evaluated before it can be trusted on a live site. This doesn't stop managers and decision-makers from asking for everything and demanding it yesterday, but it does set limits on what can be promised and what suggestions should be considered. Also, the cost of a Web project can't outstrip the value it produces, so not all goals can be met within the allotted time and budget. However, you can bring in the work of others whenever possible, which can provide additional features without additional cost. It's also helpful to use pre-existing tools to develop a Web application and delegate aspects of development to those who are most capable, including the application's intended audience.