Roger S. Pressman
Moderator

Roundtable . . .

Do the old tried and true principles of software engineering make any sense on the completely new playing field of the Internet? See what nine software leaders have to say.

Can Internet-Based Applications Be Engineered?

I will soon be celebrating my 30th anniversary in the software field. One thing I've learned along the way is that software people are slaves to fashion. A hot new technology comes along and we drop everything (or try to) to get involved with it, forgetting the lessons of the past. Internet-based software is the latest such technology.

I think our forgetfulness could get us into trouble, but maybe I'm wrong. That's why we've convened this virtual roundtable: to discuss the applicability of "older" software development ideas to the world of the Internet.

We conducted the roundtable over the Internet in June 1998. The participants come from a wide variety of software backgrounds. Some of us are old-line software engineering proponents, others focus solely on Internet-based software.

—Roger Pressman

Roger Pressman: It seems to me that just about any important product or system is worth engineering. Before you start building it, you'd better understand the problem, design a workable solution, implement it in a solid way, and test it thoroughly. You should probably also control changes to it as you work and have some mechanism for ensuring the end result's quality. Many Web developers don't argue with this; they just think their world is really different and that conventional software engineering approaches simply don't apply.

My first question, therefore, is a general one: Can Internet-based systems be engineered in the conventional sense of the term? If not, what makes them so different?

Ted Lewis: The Web world differs only in its much shorter development times and product life cycles. Instead of estimating the cost of a well-specified system, developers have been forced to estimate the quality of a fixed-time product.

Ben Adida: I agree. Web-based systems must be conceived, planned, and produced in record time. Also, because Web projects have inherently short lifetimes, why engineer a system thoroughly if next month's technology makes it completely irrelevant? A quick hack might do the trick.

The rules do have to change for Web development—you need faster tools, and you need to plan and design only as the specific project's purpose dictates. This doesn't mean that classical methods don't apply at all, or that we should build Web sites completely out of quick Perl hacks. It just means that applying solid software engineering principles to the Web often proves quite difficult. We need to adapt our methods to the speed of the Internet.

Ellen Ullman: Ben, not *all* parts of a Web application are produced in record time. The actual Web pages themselves are certainly volatile and highly disposable; speed does count on the user interface side. But many other aspects of Web applications—the organization of back-end databases, for example—represent long-term investments, which are fairly stable and slow to change.

Tom Demarco: Each time anything new comes along, its advocates trot out the line, "this changes everything, so our old methods no longer apply." There is always a germ of truth here, or it wouldn't be so seductive; new modes, tools, hardware generations, and so on do require adaptation of method. The part requiring adaptation, however, is usually small compared to what remains applicable—particularly when applying pre-Web methods to Web development. There is mostly baby here, very little bath water.

Tom Gilb: Internet-based applications should be "engineered." The top few critical qualitative requirements should be specified and a suitable architecture derived from this as well as from cost and other constraints. The system should be evolved and regularly measured against the latest update of the quantified requirements. Not doing so substantially increases the project's risk of failure, because of misunderstanding what the project is really all about.

Brent Gorda: I don't think conventional engineering skills are irrelevant to the Web. In fact, I believe the Web's accelerated develop/publish capability increases the importance of tried-and-true software engineering practices.

I've seen this shortened cycle compel programmers to bypass the (perceived) time delays of code management software and QA cycles. While programmers will claim to understand the importance of such methodologies, they justify not using them with comments like, "this software's better than the last version and we'll have new code ready tomorrow, so any bugs we find, we'll fix then."

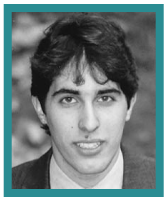
But just as it doesn't take any longer to drive with your seat belt on, solid tools and practices take no longer to use day-to-day. It's easy and human to be lazy, but the seasoned developer appreciates the value of these time-tested habits. Many such developers are working on the Web and are producing some of the better sites and tools available today.

Watts Humphrey: The engineering principles of planning before designing and designing before building have withstood every prior technology transition; they'll survive this transition as well. We had these same arguments when we moved to symbolic assemblers, high-level languages, remote computing, interactive computing, and personal computers. The real problem is that we have not done a good enough job of teaching and using sound engineering practices. Thus, many engineers have never seen how effective they can be. While we have much to learn about both this technology and how to manage it, I have seen nothing so far that threatens the basic principles of sound engineering.

Ray Johnson: Time-to-market and many other features of Web development are not unique to Web applications. While overhead is a valid concern, it's not a good reason to abandon basic engineering principles. I've seen Web application builders rely on really smart, really hard-working engineers—the brute force method. However, for projects that are too boring to attract the best engineers (most of them), this model quickly breaks down. It would be better to choose a software engineering model suited for the dynamic, fast-paced world of Web application development.

Lewis: I'll probably sound like the contrarian in this debate. "Old-fashioned software engineering" techniques that evolved out of the 1970s structured programming movement haven't worked, so why do we want to impose them on the new generation of Web designers? Today's Web programmers are trying to escape the old school by blend-

Ted Lewis is a member of the IEEE Computer Society Board of Governors, past editor-in-chief of *Computer* and *IEEE Software*, and current associate editor of *IEEE Internet Computing*. He writes for *Computer*, *IEEE Internet Computing*, *Scientific American*, and other publications. His most recent book is *The Friction-Free Economy: Marketing Strategies for a Wired World* (HarperCollins, 1997). Contact him at lewis@cs.nps.navy.mil.



Ben Adida is a graduate student at MIT studying cryptography and network security. Since 1995 he has been involved in building medium to large Web and Internet-based systems ranging from e-commerce product catalogs at the Hearst Corp. to a secure voting system at MIT. His interests include Java, security, and database-backed Web sites. Contact him at ben@mit.edu.

ing art with engineering. They know the problem is fuzzy—that is, unspecified; that their customers don't know how to specify their requirements; and that it doesn't matter anyhow, because as soon as the system is completed it will be obsolete! The waterfall model is dead, the spiral model is dying, and the rapid prototype is becoming the product. Darwinism has forced developers to adopt a free-wheeling approach to Web-based software development that makes traditional "structured" software engineers roll over in their cubicles.

Ellen Ullman is a principal of NeoLogica, which provides engineering design services to startup companies. She is the author of *Close to the Machine: Technophilia and its Discontents*. Her writings about the software engineering profession have appeared in *Harper's*, *Salon*, and in the collections *Resisting the Virtual Life* and *Wired Women*. Contact her at ullman@well.com.

Johnson: Ted, name me a "traditional" software project that wasn't a "fuzzy" problem or had a customer that could totally specify the requirements. How are Web applications so different? I agree that the waterfall method and anything else designed in the 1970s is poorly suited for current software projects. But some newer models work well for fast-paced projects.

Lewis: I violently agree with you, Ray! We still have the same basic problems in Web-centric design that we had in mainframe-centric software design. My point is that we still don't have a solution. And just because we're developing software for the Web doesn't mean we're closer to a solution.

Adida: Ted makes an important point here: the problem *is* fuzzy. We need an approach that allows for more freedom and flexibility as the project changes. It's similar to engineering F16s when all you've done before is make propeller planes. Sure, the main difference is that F16s are faster, and the basic principles of aeronautics still hold, but the engineering approach is completely different.

Ullman: Every technical generation believes its situation is unique. Web developers believe it today, and before them developers of distributed systems, client–servers, interactive systems, re-entrant code, and so on. Each generation was correct by definition: whatever they were doing was indeed different from the software "conventions" that came before. The support tools they were asked to use were always lagging, always based on the preceding technology.

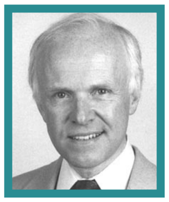
So to answer your question directly, no, Internet-based systems cannot be engineered in the "conventional" sense, but then *no* system based on emerging technology can use the previous generation's conventional engineering tools. In the business of making software, an essential tension will always exist between emergent architectures and engineering "correctness."

Speed Rules

Pressman: So we have two schools of thought here. One feels that speed of development and new technology make Web-based development different, thereby justifying a unique approach to building such systems. The other believes the need for solid engineering overshadows the "differences" in the technology being developed.

I'd argue that both views are correct, which leads to a follow-on question: How should we engineer systems that are often built by non-engineers, are dynamic, and must go live in days, not months? Stated another way, how do you advise an organization that needs to build increasingly complex Web-based systems that must work the first time and every time?

Adida: You're basically asking how to make a system infinitely scalable in days. I don't know if anyone has an all-encompassing answer. A few simple principles can help, though. First, immediately seek existing solutions. If your site needs to store information that is dynamically searchable or editable online, get yourself a relational database. For scalability, connect to it via some standard protocol (such as ODBC), so that you may simply exchange your existing small-time database for an Oracle monster in a few months. Second, separate content from functionality as much as possible. This sounds simple but is all too often ignored. Third, and most important, use a development environment that lets you move fast. Forget about writing large C programs that you have to recompile on every change. Scripting languages built into existing Web server software are the way to go.



Tom DeMarco is a principal of the Atlantic Systems Guild, a consultancy specializing in project management and litigation involving software-intensive endeavors. He has authored six books on software method and management, including *The Deadline: A Novel about Project Management* (Dorset House, 1997). Contact him at tdemarco@atlsysguild.com.

DeMarco: *All* systems need to be delivered in record time. Web products are no different, with one exception: the Web is full of glitzy eye-candy pages that convey very little real value. And yes, if you're building eye-candy, you'd better do it quickly. Such cases call for nothing more than hacking. But consider a real Web-based system such as Alta Vista or E-Trade or the FedEx dispatch center. Who could argue that such systems don't have to be engineered?

Ullman: I couldn't agree more. The front end is fluffy, and in some sense it doesn't matter what tool you use to produce it. Point-and-click baby tools are fine to generate Web-page ephemera. But everything the front end talks to is serious engineering. And since those back-end things are based on older technologies, there are indeed helpful engineering tools to support them.

Adida: Sure, these systems need to be engineered; but you can't come into a Web project you've never seen and apply all the rules by the book. You've got a new model, with a spectrum of users rarely encountered in traditional engineering projects. As for the timetable, in the Web's three years of serious existence, think of all the amazing projects we've seen. My first serious Web project, in 1995, was to build a complete database-backed, shopping-cart-enabled, magazine-selling Web site in two months—from raw machine to fully working system. That's much faster than most classical systems I've seen.

Lewis: These systems aren't "engineered" because no one uses our so-called "software engineering" in real life. Microsoft's Windows NT, for example, isn't "engineered" because Microsoft only practices one basic technique: "get a clean compile before you go home"! NT has evolved over seven or eight years of 500 people making changes that inevitably cause errors to propagate until they are stomped out by brute force. We should find out why this is a major developer's modus operandi.

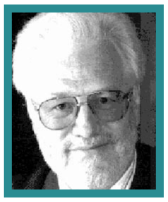
Gilb: My advice to builders of Internet-based software? One, state your quality requirements quantitatively. Two, get contractual guarantees, with penalties, for delivery and operational maintenance to those levels. Three, use proven track-record suppliers, with plenty of capital to cover problems. Four, build and expand the system in evolutionary stages. Five, consider appropriate systematic redundancy at many levels, to allow some degree of operation when failures occur.

Ullman: I know this is the "right" way to do things, but the start-up companies I work with laugh at me when I suggest something as "bureaucratic" (their word) as a simple bug list. The important thing to remember about the Web right now is that it's being created by very young people. And if you remember your own youthful ignorance and exuberance (and fearlessness), then you understand why we're even having this conversation.

Gorda: Virtually every medium to large software project I have seen risks implosion due to feature creep. To avoid this common pitfall we must recognize that software tends to grow like weeds.

My advice—especially to organizations building "increasingly complex" systems—is to keep it simple! The simpler you keep it today, the more easily your system can evolve to support tomorrow's needs. Imposing fewer needless features on your system creates fewer obstacles to supporting future needs.

Lewis: Brent, I would love to keep it simple, but how? I am an advisor to a $600 million Internet project that will take five years. How do you keep such a megaproject simple? We don't have the tools.



Thomas Gilb is a freelance consultant, teacher, and author serving clients in Europe and the US. He wrote *Principles of Software Engineering Management* and is principal author of *Software Inspection*. He specializes in software quality design and management. Contact him at gilb@acm.org.

Humphrey: The dynamics of Web-based development allow us to interactively involve users in the requirements process. This could help us quickly develop products that closely fit customer needs. When we can instantly distribute products worldwide, however, we must learn to quickly produce quality products. The current test-based approach to software quality takes too long, and will almost certainly be unacceptable in the Web-based future. Software testing can take months, and even then many defects remain.

We should learn from other fields—no other discipline uses testing as the principal way to find and fix product defects. By tracking all defects, determining their costs, and learning how to detect or prevent them, other technologies have improved quality, cut costs, and saved time.

Adida: I like that approach! I have also found that Web development forces quality into the picture very early on. In a sense, software engineering is still young compared to other types of engineering. It seems to me that Web development, with the stringent requirements it places on developers, is forcing the field to mature faster than ever.

Johnson: I disagree with you, Watts. Without testing, how do you ensure that what you've fixed is really fixed? How do you ensure that bugs are not reintroduced? Of course, you should also track bugs to determine their costs and frequency. But how can you track bugs if you don't have a test suite that looks for the most common ones?

When building a highly dynamic Web application, it's important to design and build both its infrastructure and the process for updating its content. Building the infrastructure is really just building traditional software with new tools. Building the content update process, on the other hand, is more like designing the editorial flowchart for a newspaper. Your process must have notions of data input, data validation, layout, quality assurance, and so on. This may involve a lot of human input or be largely automated. The point is that the methods you use to solve the infrastructure problem need to differ from those for solving the process problem. Neither problem is new even if their combination is unique. Breaking the problem down to well-known subproblems is one way to reuse proven engineering practices in a new domain.

Brent Gorda is president of Bonsai Software, which creates technology for use on the Web. His interests include compilers, tools, and systems. Previously he worked for Myrias Research, Lawrence Livermore National Laboratory, and the National Energy Research Supercomputer Center. Contact him at brent@bonsai.com.

Humphrey: Ray, of course you must test, but you just can't rely on testing to get quality. With the Web, it is still true that if you don't put a high-quality product into test, you won't get one out.

Lewis: The concepts preached by software engineering schools are well-intentioned, but they haven't worked largely because they do not address the practicalities of commercial development. Ray and Ellen touched on some of the reasons. I don't dispute the need, I'm simply pointing out that we lack adequate techniques. Classical software engineering isn't up to the task, so how can we foist it on the new breed of software artist? I can only suggest, given the sad state of software engineering technology, hiring exceptional people, emphasizing intrateam communication, and taking extremely small steps during development. It is still a test-intensive world.

Watts Humphrey is a fellow at Carnegie Mellon's Software Engineering Institute, where he established the Process Program, led initial development of the Software Capability Maturity Model, and introduced Software Process Assessment, Software Capability Evaluation, and, most recently, the Personal Software Process. He joined SEI after 27 years at IBM in various positions, including manager of commercial software development. Contact him at watts@sei.cmu.edu.

Ullman: My advice to organizations: Don't forget that the software belongs to the user. Web-based development has completely done away with the concept of the installed base. The software can vary endlessly moment to moment, with or without user input. But just because the code resides on your server doesn't mean that you, the developer, can keep changing it at will. When Web developers decide that old-fogey stuff like software staging and release schedules and usability testing no longer apply, they rob users of their proper sense of control over the workspace.

Young developers talk to me about a "new business model" for software, how Web software is more akin to a service paid for by the hour than a thing the user buys and owns. Even if we accept this questionable idea of users as renters, developers must remember that even landlords can't just come and go in a tenant's apartment.

The Right Stuff

Pressman: I'd like to discuss one more thing. When any new software technology or product area goes through rapid growth, "exceptional people" (as Ted called them) do the work and, by sheer force of will, produce high quality quickly. But as time passes, the workload overwhelms even the best technologists, and less competent people with (possibly) lower motivation are recruited to meet the demands of the growing marketplace. There's little doubt this will happen with Web-based software, as it has in every other software domain. How should a company handle this? How should it organize teams of people to build software for the Internet when not all of them are "exceptional"?

Lewis: Well, Roger has hit on the dilemma of most of the software industry, especially when there are thousands of job openings for programmers, designers, and technologists.

I think the first steps to salvation are for project managers to admit that (1) they really don't know the requirements, so they have to track requirements creep as a first line of defense; (2) software development teams are made of humans, not machines, so they have to track social interaction; (3) the product will be late, so they have to pad estimates; (4) there will be more bugs than anticipated, so they have to elevate testing to the level of a rock concert; and (5) there will be personnel turnover, so you have to constantly guard against hacks and inadequate documentation. I have a friend who doubled programmer productivity by taking his team to a movie every Friday afternoon! These principles are all found in Tom Demarco's book.

Gilb: We can only hope to control this chaotic environment by learning to specify the critical requirements, especially the qualitative ones, in a measurable and testable manner. Our focus, with whatever people and methods we have available, will then be to drive the project towards the testable delivery of those quality levels, evolutionarily. Inadequate people, architectures, or methods will reveal themselves quickly and will have to be replaced by stronger ones.

We also need to reward developers with progress payments for real, relevant, measurable achievements. We tend to pay for failure, and there should be no payment for failure.

Gorda: Companies faced with tight deadlines and less than exceptional people need to keep it simple and keep teams small. If necessary, they should break the task up into smaller pieces for subteams. Developers need powerful software tools and won't have time (or the capability) to build them. The tools should be chosen carefully as they will form a core piece of the solution and thus must come from a source that will stand behind them and be around in the future.

DeMarco: I believe "hacking" is at the heart of this entire debate. All the "Web is different" advocates seem to be suggesting that only hacking will serve as a method because only hacking is fast enough. Even your final question, Roger, implies that superstars do it one way (fast hacking), but "reg'ler" folks might have to do something easier—and presumably slower. This seems wrong-headed; there is nothing easier or slower than hacking. Hacking means beginning with a lousy design and flailing away at it to make it work. This is always a loser.

We need a very light process for rapid development work, whether it is Web-based or not. The heart of a good and light development process is a discipline of design: careful decomposition into cohesive pieces with thin interfaces between them. This approach can work for the superstar or the novice—the novice just needs a little more hand-holding.

Humphrey: We haven't discussed what I see as the key concern. The Web is growing and evolving instead of being designed. While it does have a designed backbone, no system design exists for multiconnected, dynamic, worldwide Web-based applications. I fear that poor design practices could introduce latent defects that will make Y2K look like child's play. Web-based systems clearly have enormous potential benefits, but what threats do they pose to a national economy that depends on the Web? We will need a trusted foundation that considers recovery, security, privacy, and protection from subversive attack. We will also need systems that fail gracefully and behave well under stress.

I doubt simple answers exist, but we will almost certainly need standards and engineering disciplines. The future of Web technology is exciting, but I suspect that professional practices will be more rather than less important.

Ray Johnson is engineering manager for Scriptics, a start-up company focusing on tools for the Tcl scripting language. His interests include programming languages, GUI toolkits, and Internet agents and mobile code. Prior to joining Scriptics, Ray was a researcher at Sun Microsystems Research Laboratories. Contact him at rjohnson@scriptics.com.

Johnson: Fred Brooks, in *The Mythical Man-Month*, points out that the best programmers are almost an order of magnitude better than poor programmers. Twenty years later this still rings true. But if your "exceptional" people do not produce a system that is easy to maintain, you may find yourself starting from scratch when your star engineer gets a new job. We must remember that software tends to hang around much longer than engineers. The cost of having your best engineers follow basic engineering principles, then, would be dwarfed by that of having to rebuild your system because the people who wrote the code are no longer around.

Ullman: I know you're expressing the received wisdom, Roger, but I strongly disagree with your premise. Every person you hire is (or can be) exceptional for the situation at hand. After the initial development phase, you need a very different sort of exceptional person. When working code exists, "sheer force of will" is a destructive quality, in my opinion. The truly exceptional skills I look for in later-phase work are the ability to communicate with colleagues (while also being a good programmer, a rare enough combination), the ability to read and understand the system developers' "brilliant" code (ability to read code is so rare that it's extra-exceptional, in my experience), and so on. A whole new complex of exceptional skills comes into play when a project matures. It's foolish to think less of your next cohort of hires. It's self-defeating—if you're looking for ordinary people, that's exactly what you'll get.

Adida: You've indirectly pointed out one clear danger of the idea some of us expressed here: while we need a new engineering model for the Web, completely sidestepping the engineering process is a truly bad idea. In fact, some Web projects may be more susceptible than other systems to the kind of degeneration you describe.

The solution here consists in going back to the basics: serious, complete documentation. Modular design. Planning for future updates. Of course, many Web projects remain inherently short-lived. For those, even though it may break a program manager's heart, it's okay to plan for few if any updates, and thus to never consider the consequences of hiring "less qualified" people. The difficulty lies, of course, in determining which projects will be short-lived. That you can leave to the "exceptional" people at the start of the project.

The Final Word

Pressman: As moderator, I've reserved the last word for myself. In listening to your collective comments, I get a feeling of déja vu—we've had this discussion before, in another time and focused on other technologies, but with the same basic arguments.

I submit that engineering discipline is *never* a bad idea. The basic principles that lead to high-quality systems apply whether you're building the latest and greatest Web application or the 3,000th version of a corporate payroll system. The problem isn't in applying the principles of solid software engineering—it's in the overly dogmatic, bureaucratic application of the engineering process.

But that problem contains another one. Stated simply, what's dogmatic and bureaucratic to you might be perfectly acceptable to me. Every organization has to make decisions about the proper application of engineering principles as it builds Internet-based applications. And that's okay—in fact, it's really a business decision. But if an organization chooses to ignore software engineering altogether, I think both managers and technologists are making a mistake that might come back to haunt them (or their replacements) long after the excitement of the Web has waned. ❖

About the Moderator

Roger Pressman has worked as a software engineer, manager, professor, author, and consultant, focusing on software engineering issues. He has authored six books and is editor of *IEEE Software*'s Manager column. His book *Software Engineering: A Practitioner's Approach* is a widely used software engineering textbook. Readers may contact him at pressman@rspa.com, or via the Web at http://www.rspa.com.