Presentation
Paper
Bio
Return to Main Menu

P R E S E N T A T I O N

W14

Wednesday, February 14, 2001
1:30PM

THE WORLD WILD WEB: A NEW PARADIGM OF RESPONSIVENESS & RELIABILITY

Jenny Jones
Segue Software

International Conference On
Software Management & Applications of Software Measurement
February 12-16, 2001
San Diego, CA, USA

The World *Wild* Web: A New Paradigm for Deploying Web Sites

Jenny Jones

Segue Software

Who can use this information?

This talk is about the methodology of web site deployment, and is directed at:

- **Business Development Managers**
- **Project Managers, Development Managers**
- **IT Professionals, Programmers, Testers**

Anyone who needs to understand the risks involved in having and/or building an eCommerce system

What are we going to talk about?

Changing the standard deployment paradigm

- **Why is the world different now?**
  - Old World of Expectations
  - Old Software Release Methodology
- **New World of Expectations for the Web**
- **New Release Methodology**
  - Site Maps
  - User Profiles
  - How is the whole effort managed?

Why is the world different for the web?

A presence on the Internet exposes the inner workings of an organization magnifying both its *strengths* and *weaknesses*.

How can this happen?

- To the outside world, ***your web site is your business***. Customers expect the same level of service from a web site as they would from a brick and mortar store.
- To you, your web site is just one more project to manage. It is easy to lose sight of the criticality of customer perception and satisfaction.

Internet *exposure* can *expose* weaknesses

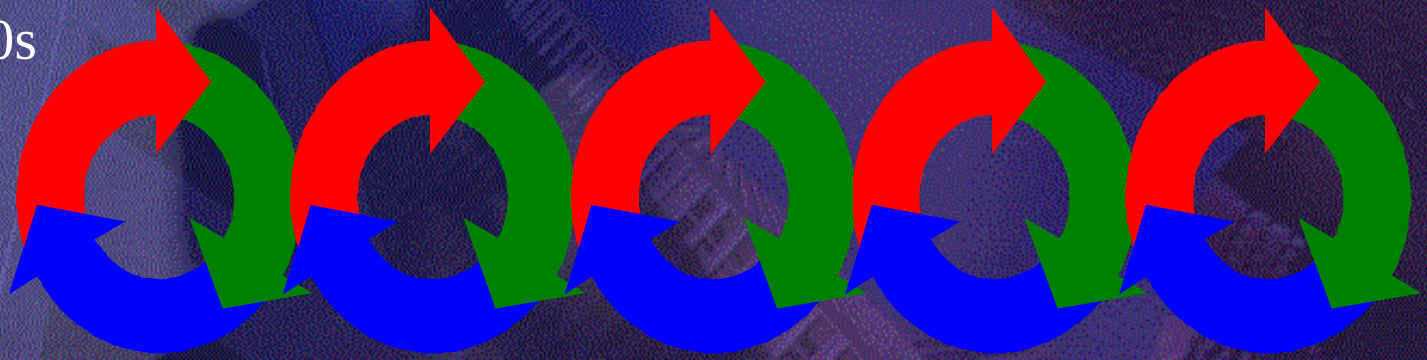
How are weaknesses exposed and magnified?

- Web users have a high performance expectation: customer demand for a fluid coupling of speed and high-level functionality is hard to satisfy…
- Content is not reviewed frequently enough, and customers see through this immediately...
- Release cycles shorten tremendously: no sooner is a web site “up” than it needs updating. Development teams face an unending struggle to even “finish” site updates...

The deployment cycle has changed...

1990s

2000s



The old days...

The standard product deployment cycle was guided by a long-term vision of a product's future

- subsequent releases added more and more functionality to support that vision…
- testing phases added more and more scope to testing efforts as the product matured…
- the long-term view was the goal: each release was a step in that direction, rather than a thorough development of individually functional pieces…

Why did we do it this way?

We believed *(and the market supported…)*

- **innovators want power not convenience**
- **innovators are *the initial market segment***
- **new software = high expectations for previously unavailable features and functionality**
- **new software = low expectations for usability**

*This meant **time** for the development team to improve the product as a whole!*

The new days...

The truth about customer expectations:

The average Web buyer will wait about 8 seconds to load a page before attempting to find another web vendor, *and it's getting shorter!* *- Zona Research*

The truth about our web sites:

The *average* web page takes almost 10 seconds to download, *and it's getting longer!* *- Zona Research*

The new ways…

"If we have downtime or are slow, customers go away and some of them don't come back."

- Steve Gold, VP FreeEDGAR

"If we don't perform well, we risk something much more important than [today's] dollars: our customers' good will…It's very expensive and very fragile.

- Dwight Gibbs, CTO Motley Fool

The new ways...

And it's not just a matter of response time:

The cost of site downtime is even higher. Forrester Research puts the average cost of site downtime at about **$8,000 her hour**. Pure eBusinesses like Amazon.com or eBay have a **much higher cost**.

eBay's 22 hour crash: $5 million

ESPN's Fantasy Baseball crash: $7.8 million

What does this all really mean?

In the Development World:

- *We are accustomed to adding new functionality* even though it may be buggy or hard to use…
- *We are not accustomed to perfection* first time out…
- *We are not accustomed to flashy interface changes* such as those the web market demands on a regular basis…

In the web world, users don't tolerate bugs or inconvenience...

Perspectives on the new ways...

- **Planner's Perspective:** eBusiness is the greatest opportunity *and* the greatest threat my business has ever faced…
- **Customer's Perspective:** if one place doesn't have it, or can't deliver it *now*, some other place *does* and *will*, and I don't have to leave my desk to find it…
- **Developer's View:** everything in the specs has to work right the *first time* and *every time*, for *everyone*…

How am I supposed to do all that??

Welcome to the World of Engineering!

Classical engineers have been waiting for the day that software engineering in all its undisciplined glory finally gets its comeuppance.

That day has arrived with the *World Wild Web*!

What's wrong exactly and how do we fix it?

- Standard Software Product Deployment has always assumed that usability is secondary to new functionality…
- On the web, usability expectations are the first priority of every customer. At the same time, newness is demanded: new stuff, better stuff, constant progress…

The priorities of classical engineering return: reliability comes first!

Get out your CAD software!

To develop and deploy our site to live up to market expectations, we must…

- **Define user profiles** that represent all the flows in and out of our system…
- **Create a site map** that shows the structure of the system and where the components are housed…
- **Map the user profiles onto the site** so that we understand the flows of our entire system and how different flows interact with one another…

What are *user profiles?*

- ***User profiles** are collections of activities that can be executed on your site.*

  Because we normally talk about *who* is doing particular activities, bundling collections of activities based on target users is an effective way to define and categorize all activities you wish to enable on your site.

- **Golden Rule: if an activity doesn't support the needs of at least one of your target customer groups, it isn't worth doing.**

What's a *site map?*

Web Site Maps for Web Users:

- Information and activity content of a site displayed in some standard order (such as alphabetized topics)

Web Site Maps for the Development Team:

- Functional view of major components
- Machine mapping to the functional pieces
- Arrows, lines and other flows that map activities of users in and out of functional areas and in and out of different machines

Examples...

If these are the keys to running our web efforts, why haven't you given us any pictures?

Site Maps and User Profiles are as specific to your site as fixtures, appliances and family members are to your house…

Getting started: user profiles

- **Write out profiles for all groups of targeted users you expect will come to your site:**
  - Surfers – casual readers
  - Prospectives – people interested in becoming customers
  - Customers
  - Employees

- **Write out lists of activities wanted or needed by each group…**
  - Notice the overlap: evaluators, for instance, will engage in some of the same activities as established customers…

The list of activities you generate from these profiles is a checklist of functions that must operate correctly and quickly all the time.

Getting started: site maps

- **Map out the machine you know you will be using to support the web site**
  - keep a database of all machine characteristics: OS, software, versions, resources, etc.
- **Map the functional components to the machines**
  - the database is on which machine?
  - the web server is on which machine?
  - Keep a database of all component characteristics: version, third-party integration, incompatibilities and limitations, etc.

Getting started: map the users to the site

Understand how your system works: for every profile, map the flow of each activity through the system of machines and functional components.

- **The site map and the user profiles represent the pieces of the puzzle that persist and must remain reliable from "release" to "release" of the site.**
- **All design and testing activities should revolve around the stability of these critical components.**

Wrap up...

What's going to get your web team through this new world?

- ***You know what's different about the new world:***
  - **User expectations are high!**
  - **No time in the release cycle!**
- ***You know why the old model doesn't work:***
  - **Usability was second, not first!**
- ***You know what you have to do:***
  - **Organize your design and deployment strategy around:** ***User Profiles Site Maps***

The World *Wild* Web: A New Paradigm of Responsiveness & Reliability for Designing and Deploying web sites

Jenny Jones
Director of Product Services
Segue Software
Lexington, Massachusetts
jjones@segue.com

Abstract

Computer-related businesses, organizations, and institutions have universally recognized the need for visibility on the World Wide Web. But what used to be a simple tool for educational institutions to share information has turned into a new venue for marketing, selling and gathering information. This effort requires a design and deployment plan that is fundamentally different from all other software products or projects ever conceived because of the extremely wide accessibility and expectations from target audiences. This means that software organizations face a paradigm shift in order to adequately address the radically new needs represented in the World *Wild* Web.

Most software organizations have their own release process deeply rooted in a long-term cost model where one basic assumption is made: as the releases roll and the software "matures", the design and testing cycle, will (paradoxically, it seems) grow shorter and more extensive as time goes on. What this means is that active design work is plugged into an existing model and active testing incorporates existing regressions and new testing. In both cases, reuse of past work will allow the body of work to grow, but each iteration of adding to that body of work will grow shorter and shorter. This model is based on one very important assumption: the software interface is a part of the system that changes very little. This model doesn't fit the web community expectation of fast-paced site updates with a constant flow of newer and "better" ways to get and give information. These newer and better ways invariably translate into a changing user interface -- the web pages. Without a stable interface of unchanging pages, traditional software development strategies fall apart.

Why should you consider changing the engineering style for your organization's web site?

We often *assume* -- particularly for new product ideas -- that the target customer base will tolerate a certain amount of poor performance or inconvenience in using a new product if a few key features that no one else has are available. This assumption isn't a poor one by any means. At the onset, typical markets/customer bases include a segment of so-called innovators. This group is comprised of seasoned professionals that will take on new technology if it gets them information or functionality they couldn't otherwise get without some surmounting difficulty. Most software projects hit this segment first, and work to improve usability as a part of the software maturation process. But the web site is not a typical software product, although the workflow for creating and updating the site goes through software engineering teams. A web site must not only push its content and pull in visitors. Without exception, the site must do so *without fail and without surprises.*

In this way, a web site has additional parallels with a traditional store. In the brick-and-mortar world, window dressings and displays are constantly changing. Specific merchandise moves out. Stores have sales. On the internet, front pages constantly change. Inside formats of pages change, while content moves and evolves. Notices go up. And for online retailers, sites have sales. With a store, you can intuit the basic, underlying structure that doesn't change. You can imagine how one might build a store and make sure that it functions without needing to check every window display or sale process. The same sort of paradigm can apply to designing and testing a web site: there will be an underlying structure of activity and technology that will persist throughout all the cycles of changes.

This is very different from what software teams are accustomed to producing. Software teams are accustomed to putting out rough interfaces that accompany some powerful process under the hood and make usability something for following releases. On the web and with few exceptions, if you cannot make it usable from the beginning, then you better not put it on the web site. Simple is truly better because complexity puts a high risk on failure. The more complex the transactions are over your site, the more risk there is of slow responses and downtime.

There are very big costs in terms of opportunities and real time costs generated by site downtime, poor functionality, and by generally not meeting visitor expectations. eBay's highly publicized 22-hour crash in June 1999 is reputed to have cost the company more than $5 million in returned auction fees. ESPN, which lost its fantasy baseball site for three days beginning July 11, 1999, conceded that it will have to compensate some of its 260,000 online players, who pay $30 each to play in the league, at an approximate total of $7.8 million. In both cases, more *up front design, performance testing* and *ongoing integrity testing* would have revealed the configuration and design bottlenecks that caused these crashes. But there are more subtle, more common, and more important risks than something as large a scale as a crash that brings a site down entirely: annoying the finicky consumer (we all know how much it costs to get a customer back).

- "If we have downtime or are slow, customers go away, and some of them don't come back," said Steve Gold, vice president of FreeEDGAR.

- "If we don't perform well, we risk something much more important than dollars: our customers' goodwill…It's very expensive and very fragile." Dwight Gibbs, chief technologist at the Motley Fool financial information site.

Zona Research reports that the average Web buyer will wait about eight seconds for a page to download before looking for alternative sites. As the internet world grows and matures, that number can only decrease for user expectations for response time will grow shorter. In 1999, Zona sited an average download time across a backbone connection as 9.93 seconds, longer than the eight-second threshold for the impatient consumer. That means it is critical that response times for your site are probably longer than the average web user will tolerate. But the high expectations don't stop there.

Slow response is only half the battle. The cost of having genuine site problems where user cannot get access is even higher. Forrester Research puts the average cost of site downtime at about $8,000 per hour. "Pure" eBusiness sites like Amazon.com or eBay have a much larger cost. For example, eBay's 22 hour crash equated to around $23,000 per hour. Moreover, this eight-second threshold speaks very clearly that the site doesn't have to crash to turn away business. If the customer simply has to *wait* (for whatever good reason we as the web site producers might have) the customer will look elsewhere. That's a kind of usage demand that software engineering projects have never had to come close to meeting. [1]

What does this mean for developing web technology?

eBusiness has created a wild new world of software deployment with new product specifications driven by the customer and changing at a rate never considered in any engineering project before. Companies as large as Fidelity have front-end pages that change as frequently as weekly. There are no generalized testing methodology courses, no industry-wide best practices, no technology nor architecture standards for this new eBusiness world. Customers expect flash, freebies, newness, and speed. Web Developers expect maximum room for creativity and quick turnaround for testing. Everyone expects tremendous performance integrity with no downtime and responses in the milliseconds. In addition to a radically different development standard and incredible end user demands, IT Departments (ultimately, a hidden portion of your team whose contribution is vital to the "success" of your site) have a plethora of choices for machine configurations. On any given day, IT can change environment settings for databases, web servers, clients, connections, and many other areas that affect the system and functional performance of your web site.

The Planner's View of eBusiness: "…eBusiness either represents the greatest opportunity for your business or the greatest threat to its success." [2] This characterization gives eBusiness systems first priority on the schedules of all web-aware CIO's and IT executives. This inside, top-level visibility adds to the pressure to perform in a time frame never before considered.

The Customer's View of eBusiness: Because of an unprecedented availability of alternatives, customers won't wait for your site to respond and they won't wait for you to fix "bugs" in your online process. This outside, front-line connection to your revenue or information stream means that no bugs in

your primary processes will be tolerated whether you like it or not.

The Developer's View of eBusiness: The software engineering process has an old paradigm that isn't geared towards a changing front-end nor towards a usability-first, functionality-last development model. Marketing Requirements Docs, Product Specs, Design and Functional Specs are all most frequently addressed through front-end primitives -- the basic GUIs for an application. Standard development and testing cycles revolve around a stable GUI. In this methodology, basic hardware and software configurations are assumed from the onset of the development process. The end users are controlled through the flow of activity at design time, rather than viewed as the group that defines activity.

The old release model looked past an upcoming release to the planned evolution of a system. The primary objective of designing and testing was to speed up deployment for each successive release of an application or system. There is an implicit bottom-up approach to producing software here: production and test of a component, integration among related components, creation of a hierarchy based on minimal interaction between components, and finally an end-to-end integration test of all components. Sometimes at the very end of the release cycle, we make a cursory attempt to check interoperability with outside systems/software. This means that the release model most software teams employ does not put usability first, cannot accommodate a changing interface, and does not incorporate flexibility for end user systems or production systems.

The web turns this methodology inside out. On a web site, all the major activities for a user are displayed at all times. No window or access hierarchy protects the system from any flows outside the scope of anticipated, hence designed, activity. As a result of this basic lack of well-defined and structured activity flow and the sheer pace of expected updates, we must create a new methodology: a *simulation* approach that focuses on the *reliability* of an impending release. In other words, design and testing efforts for the web focus around prioritizing to maximize reliability, then simulating online processes to verify that reliability, rather than running the traditional model where design is complete and testing is an iterative process of catching more and more.

The goal of this approach is ensure that a site's most critical processes function and perform to expectation on a release-by-release basis. Functionality that hasn't been thoroughly tested for usability is not incorporated into a public release. New functionality that anticipates a great increase in traffic on the site is not added until the software team verifies that the current configuration can handle the additional load without downtime or great increase in overall response times. This approach is top-down, enumerating and prioritizing functions in relation to impact on reliability. You turn those functions into transactions (i.e. functions that can be performed at any point in the use of a site) and create your design and testing plan from performing these basic tasks through the eyes and access privileges of the target users. In conjunction with this activity-based, transaction testing scheme, you also form a battery of operability standards. These standards form a base of tests aimed at catching obvious defects that, because of their simplicity, would reflect negatively on the business the site is representing.

Like a single typo in a resume, even a small imperfection in reliability for your web site's most visible areas can destroy the effectiveness of the whole site. The most visible areas must be perfectly reliable for the first customer who encounters them and for every customer after that. Perfection from the onset is necessary on the web. New pieces are not put into the public domain until they are consistent and reliable. That means that both clear articulation *and* prioritization of the critical functions are necessary to organize all design, test, and deployment efforts.

How do you organize this effort if traditional methods are not effective?

Traditional methods loose applicability because they do not map to the parts of the web site that stay the same. All of the front-end change that any given web site undergoes over time epitomizes the antithetical nature of the relationship between producing reliable, quality web sites and the standard software product development cycle. The assumptions and conditions for using a standard product development process cannot coexist with the highly variable conditions for running web sites. This makes applying traditional release methods at any point in the production cycle either an incredible nightmare or simply something that is not done. The concepts of *site maps and user profiles* are the pieces of the puzzle that do persist from "release" to "release" of a web site. These two tools can be used to form a new way for deploying a site.

The site map, the first organizing component, is a multi-dimensional representation of the activity flows the web site is supposed to facilitate and the connections among those activities. You can manage all phases of the web site release cycle -- design, implementation, test, deployment, and ongoing

updates -- using this primary information organization tool. The site map identifies all the logical paths that the user can navigate through on the web site. It describes all paths that you've created, and can be used to define or refine the user profiles (which we will discuss later) you create for the target audiences that you expect or want to use the site. The site map should go down to the page or CGI level of the site so that testers have a map of how the application works. The site map is something akin to an architecture map or specification of a system, except that site maps by nature will have logic branches that change frequently, while overall components or areas of interest will not. In the client-server world, such a map usually isn't produced because system changes happen at a much lower frequency. Often then, transitional teams build an implicit shared map together that's never fully or formally outlined. The pace and nature of changes on a web site make an explicit map essential for the web world.

What are the Site Map components?

- *Functional representation of the main components of the site.* This is a flow chart of all the activities that a user can initiate on your site. An accompanying document to this chart should include specifications for CGI scripts or any executables that take input and produce web-page content as output. You will need those to build tests that have flexible input/output parameters but represent the unchanging structure of the web site.

- *Machine information and mapping to main components each machine will carry.* Software and hardware configurations have never been more important to software projects as they are for web sites. IT Departments have a plethora of choices in a multitude of levels that can affect site performance and functionality. Mapping this accurately and keeping that map up to date are essential for pinpointing problems in your system.

- *Software/technology information of the machines that manage, house, or interact with functional parts of the web site.* This is where you clearly note and religiously update those notes that detail your server software and hardware configurations. It is essential to write down the library and tool kit versions the development team is using to build the web software as well as the library and tool kit version that MIS is supporting.

- *Representations for various users that will hit the web site.* Users will have different categories of privileges that change the process that's exposed to them. Mapping the different user views of the web site is the only concrete way to track the effect and risk of proposed changes without having to know the construction of the entire site. The word "view" is not used arbitrarily here. The user views have a direct analogy to the view counterpart in the database world. A view takes an entire structure and highlights particular paths and connections for that particular view.

This mapping method may seem complicated, but in the end it represents the parts of the system that are most likely to persist for many cycles of site updates. It also creates a forum for targeting and prioritizing the actual activities that the site is designed to cover. All initial efforts to create the site map and educate the R&D staff of its importance and use will be well rewarded.

Like a mission statement, the site map guides and aligns the entire team of people needed to launch and maintain your web site. An accurate site map will draw attention to design bottlenecks as well as component or machine bottlenecks, if you keep the machine specs updated. The site map also points out gross design flaws very early in the initial or ongoing release cycle -- for instance, when a critical process like online purchasing is hidden from the front page.

In this sense, the web offers the first opportunity to truly prioritize the functions of a software project. Usually we have a large pile of considerations, with a somewhat arbitrary process of prioritizing the list. In this case, because the room for even marginally missing customer expectations for speed and ease of use is nonexistent, the prioritization becomes clearer. Pick a handful of activities and implement them with top notch usability standards. If you don't finish the entire look and feel of the user experience for a particular process, you need to drop it off your list of processes to webify. The site map, as it name implies, maps out the system. User profiles guide your design and testing efforts through that system.

User Profiles, the second organizing component, define how you will navigate your site, set the mark for expected volume of use and ultimately define your design and testing efforts. The components from the site map should start out as the basic set of activities you want the web site to perform at the highest level, for example, research technical information, purchase, browse offerings, update customer records. Each user profile will then access different combinations of components, pages, and controls of the site. Different classes of users will have very different loads on the system.

For example, surfers who hit your home page and a few second-level links will have little impact on performance despite high numbers. In contrast, a relatively small number of customers logging into your site and each conducting more involved transactions that simply surfing (like purchasing or updating records) can have a significant impact on your web server. You need to make explicit lists of users for sites and expected activities -- do not define these lists page by page, rather start out with a user definition and a general list of activities. Each profile should correspond to a general activity set that your site supports. Examples of common profiles are these:

- *Surfer:* someone who loads the home page, hits a few links, and leaves.
- *Customer:* someone who engages in information gathering or whom your business wants to capture information about.
- *Buyer*: someone who purchases from your site, if you offer online sales.

All three of these groups will want different functionality from the site, and all three profiles will generate a different quality and quantity of traffic on your site. Surfers, for example, will want to have all information available to them a maximum of two clicks away. Their activity will most likely not be process-heavy, unlike a buyer or customer who might interact with a database would, but their class will probably generate the most traffic of static pages, if you site is designed to pull visitors to it.

Customers, on the other hand, will want some information exchange and will likely interact with a backend database or with CGI systems that do some processing. There will be fewer customers than surfers. Buyers will be the smallest class of the three, but will have the most resource-intensive activities because any monetary transaction will have several security layers, database logging and record retrieval.

In the old world where applications ran directly on the operating systems, a designer or tester could navigate the system from a GUI, basing tests on the mapping of the window hierarchy. Because the web browsers give you the ability to interrupt the flow of any set of web pages through which one can navigate, an implicit hierarchy no longer exists. Rather, the navigation of a system is solely based on assumptions made at design time about *who* is going to use the site. Very little attention is paid to how the user might get to a certain place. This lack of application access control urgently underscores the need to first define, than consistently review, prioritize and test the site around what target users. The user profiles define activities and access. The site map documents how those users navigate the system to execute those activities. Management prioritizes the activities on an ongoing basis to ensure that business objectives are met without risking visitor perception about the site's reliability.

Your task must be to map out and verify that persistent system, that structure which defines your web site. That structure must work hand-in-glove with the short list of critical tasks that must function perfectly and consistently on your site all day, every day. You can map the backbone and the critical functions effectively, efficiently and easily by creating and updating your site map and user profiles.

The important thing about the site maps and user profiles is that they remain accurate. Software projects need to have a grand design, a vision, and all that good stuff. But the execution model for a web site must be based on the reliability of what is being used/released *in the present or immediate future*.

Conclusions

The paradigm shift is in the method of design and execution, not in the basic motivation of a release strategy. In other venues, the GUI is the organizing principle for design and prototype, and hence is assumed to be one of the most stable parts of a non-web piece of software (often, yes, this is not a valid assumption, but it is a common assumption, regardless). So automation, and testing in general, revolved around stable portions of the system. That still holds true in the web world, but that what parts of the system are stable and get tested and how those parts are tested differs. In the web world, *site maps and user profiles* are the pieces of the puzzle that *persist* from "release" to "release". They are the major components of *the big picture* that your organization creates when adding web site accessibility to the business plan, and ultimately *form the measures for success* of your web site.

Use site maps and user profiles to plan, guide, test, and measure your web site effort. These two tools will be instrumental in both your functional testing and performance testing. Site maps and user profiles will be the guides you use for deciding what you *must* test.

Acknowledgements

The author would like to thank the Segue Software Gain eConfidence writing team for providing ideas and guidance.

Bibliography

[1] "The Cost of Downtime" by Tim Wilson, Internet weekly Online, Friday, July 30, 1999.

[2] Gain eConfidence, Segue Software, page 1.

Jenny Jones

Jenny Jones is currently the director of product services at Segue Software in Lexington, Massachusetts, where she acts as an industry expert writing and speaking about how software teams and business heads need to address the web, as well as running a small web project around Segue's product knowledge base. She moved into software engineering six years ago after completing her Master's in Mathematics. She describes herself as "an education junkie" and enrolled in classes at night almost as soon as she left the halls of academia. She will receive her MBA from Boston College in 2001.

Her commentary and writing is included in magazines and books including the upcoming *eCommerce Testing* by Addison Wesley and *Automated Testing Toolkit* by John Wiley & Sons (both due out this summer), and in industry journals including "eCommerce Business".