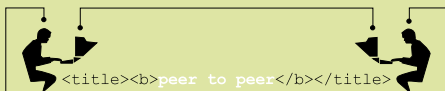




ion technology newsletter...an information technology newsletter...an information technology newsletter...an information techn
.number 1...summer 2002...volume 5...number 1...summer 2002...volume 5...number 1...summer 2002...volume 5...number 1...summer





peer to peer is published by NOLA Computer Services, Inc. to recognize the accomplishments of our NOLA team members and to provide our customers with information concerning developments in the field of information technology.

NOLA offers a broad range of information services to companies and organizations in the public and private sectors. Our services include information management consulting, system architecture, and project management.

For inquiries, please contact us at one of the locations shown below, call toll free at (888) 488-1101, or visit our website at www.nolacom.com.

peer to peer wishes to thank Chet Hendrickson for generously allowing us to reprint his article, "When is it not XP?"

Trish Thomas edits and manages the publication of peer to peer. Trish Thomas and Bonnie Bart designed this issue. Syed Baber, Danielle DeAses, Bill Gough, Joseph Graves, and Narti Kitiyakara contributed to this issue.

We welcome your comments at peertopeer@nolacom.com.

www.nolacom.com

Corporate Headquarters
3535 Canal Street
New Orleans, Louisiana 70119
Voice: (504) 488-1111
Facsimile: (504) 488-9955

2100 W. Loop South, Suite 900
Houston, Texas 77027
Voice: (713) 590-3616
Facsimile: (713) 590-3619

Five Concourse Parkway, Suite 3100
Atlanta, Georgia 30328
Voice: (770) 804-5933
Facsimile: (770) 804-5801

1800 Diagonal Road, Suite 600
Alexandria, Virginia 22314
Voice: (703) 684-4454
Facsimile: (703) 548-9446



iso 9001 registered

changing faces

Peer to Peer is getting a new look! Keep an eye out for new features in our next issue.

In this issue, Bill Gough tackles the question of quality: how we define it, how we measure it, how we make it a vital part of our business strategy. Beginning with the next issue, Andrea Washington will be reporting on ISO developments at NOLA.

Narti Kitiyakara compares the performance of different technologies when decoupling Web pages in this first article of a three-part series, page 4.

Chet Hendrickson asks, "When is it not XP?" and describes the key principles of eXtreme Programming on page 6. Mr. Hendrickson helped to conceive this agile programming methodology even before the term eXtreme Programming was coined. Now he helps other software teams improve their development process using XP's core values.

Version control is an integral part of any development project, and a version control system is the best way to keep up. Read about the beginnings of version control and the different systems available today in Joseph Graves's article on page 8.

Try to solve the new puzzler from Syed Baber on page 9! Remember, if you send in the correct answer by September 15 you'll be eligible for the drawing for a free lunch.

We aren't kidding around! We really do welcome contributions from all NOLA team members. Why write? Besides the opportunity to share your ideas and expertise with your colleagues, there is an added bonus: we are introducing honoraria for contributing writers. Contact Trish Thomas for guidelines at pthomas@nolacom.com.

quality. the final frontier.



billgough

These are the voyages of the NOLA enterprise. Its ongoing mission—to explore strange new policies and procedures, to seek out documents and records, to audit till the cows come home, to boldly go where no company has gone before.

Quality. What does it mean? What does it mean to you? And how do you know that quality work has been done? Robert Pirsig pursued this question in *Zen and the Art of Motorcycle Maintenance*. At one point, his conclusion was that you can't define quality, you can't wrap your arms around it, but if you reviewed say, several essays or source code listings, odds are good that you would be able to pick out the quality work from the substandard work.

That's all fine and good for a novel that discusses philosophy, but how does that idea translate into our day-to-day working lives? Well, if you look at the Star Trek rip-off above, you could say that quality is defined and controlled by documentation, recordkeeping, and audits. And, to some extent, you would be right. After all, that's why the auditing requirements of ISO 9001 are set up the way they are. You periodically review the work that's being done and determine whether or not the company is doing good work. The work may have deviated from the prescribed way of doing things, but it's possible that these deviations were good and beneficial, and truly represent quality, moreso than what was previously defined.

We held several internal audits this year and recently our registrar, BVQI, conducted a surveillance audit. We looked at the deviations from documented practice and determined whether those deviations were good. In many cases they were. You need a starting place, and that starting place

is our documented policies and procedures. Sure, they aren't perfect, but that's a part of maturing a quality system. We learn what works and what doesn't work, and we document and support good practices while discarding bad ones.

Over time, the feel for what constitutes quality work will change. The 2000 revision of ISO 9001 is guaranteed to change the definition of quality work. The 2000 revision shifts to a process-based focus as

opposed to 20 distinct activities. It also introduces continual improvement and better integration with business strategy and objectives.

Central to the changing feel for quality work is continual improvement. In a nutshell, it says that—no matter where you are—you can improve something and you should align your organization to accomplish that improvement. As a result, we will always be pushing the boundaries we have set for ourselves.

This is Bill Gough's last contribution to **peer to peer**. He's moved on to Albany, New York, where he will begin graduate studies in the fall. To his friends and colleagues in New Orleans he says, "thank you for being there, whether with a laugh, a willing ear, or a constructive critique. I have been enriched by you, and I hope I left you something in return. You have my warmest wishes for continued health and prosperity. God bless."



© 2001 The New Yorker Collection from cartoonbank.com. All Rights Reserved.

"On the Internet, nobody knows you're a dog."

3

webpage decoupling

nartikitiyakara

In my last article, I discussed the concepts of cohesion and coupling. Here I begin a series of articles demonstrating the ways in which you can achieve the goals of high cohesion and low coupling when creating dynamically generated web pages. I'll also look at some of the different technologies used to generate Web pages. To do this, I'm going to take a small subset of requirements from a larger project, implement them in a simple Web page using Active Server Pages, and talk about how ASP compares to Java Server Pages. In the next article, I'll implement the same requirements using Microsoft's Internet Server API and JSP custom tags, both of which allow us to create extensions for our Web servers. This will show how each technology helps us decouple our presentation from our business logic. The final article will explore the use of XML to completely separate business logic from data presentation.

Requirements

Let's say we're working on a prototype for a system that maintains résumés. Other teams are working on other parts of the system, but it's our problem to implement the portion of the system that maintains educational information. We must support five basic requirements:

- ✓ Given an HTTP request containing a parameter named `personID`, display all of the educational items for that person.
- ✓ Allow the user to edit any item on the list.
- ✓ Allow the user to add items to the list.
- ✓ Allow the user to delete items from the list.
- ✓ Must run in Microsoft's Internet Information Server 4.0 or better.

The table in which the information is stored is called `T_EducationHistory`. See sample *record* below.

Implementation

Both ASP and JSP allow us to quickly code the requirements that we've been given, and—architecturally—there is very little difference between the two. The former happens to use Visual Basic script to control the flow of HTML on a page, while the latter uses Java. Big deal. Another difference between the two is that ASP is interpreted every time the page is requested, while JSP is compiled the first time it's requested and then serves every subsequent request from the compiled version. This can make a difference in performance, but on simple pages like ours it's not enough to worry about. Alas, neither technology really lends itself to the highly decoupled design that we'd like to achieve. In fact, you might argue that both ASP & JSP make it *too easy* to produce an external coupling between the HTML presentation and the business logic that controls it, because both technologies almost *encourage* you to mix

HTML with VB/Java code.

For this example I'm going to use ASP, because I know that it only needs to run in IIS. You can run JSP in IIS with third-party IIS extensions, but when you don't need cross-platform compatibility ASP is often the more logical choice. So, having chosen our platform, let's get started. Listing 1 (on page 5) shows the static layout of our page with some sample data.

Listing 1 seems about right. The problem now becomes changing it from a static page to a dynamic listing. To work with the education listing, I'll put it into an ADO Recordset. Although Recordsets are most often associated with databases, they can be used for any data source. I'll also decouple the presentation from the database access by putting the actual code for getting the listing into its own subroutine. We'll look at the code for getting the listing shortly; in the meantime, let's look at how it's used in listing 2, which replaces the text inside the `<TBODY></TBODY>` tag in listing 1.

The problem with this code is that it puts some of the VB script into the HTML. Unfortunately, neither ASP nor JSP give us much choice about this. I could

record	field name	data type	description
	itemID	Numeric	The primary key for this table. The value is maintained by the system, so we don't need to worry about it.
	personID	Numeric	The foreign key used to relate this table to the rest of the system.
	institutionNM	Varchar	The name of the institution at which the person studied.
	degreeNM	Varchar	The degree that the person achieved.
	yearNUM	Numeric	The year in which the degree was achieved. This must always be between 1940 and the current year (inclusive).

put the whole loop into a method, but this would make it harder to see what's happening with the HTML.

Besides, if I simply moved the loop into a method as is, I'd just be moving the external coupling from the HTML body to the method. I could make the HTML a parameter to the method and mark the positions for the database values with some special sequence of characters, but that would be overkill for a simple application like this. So I compromise on the external coupling with the loop, but not on the coupling of the database or indeed on the way I get information from the HTTP request. Listing 3 shows the two functions used by listing 2. Our application will now display a listing of education items.

Our next requirement is allowing the user to edit an item on the list. The customer has indicated that he wants to activate edit mode by clicking on a link associated with each line item in the listing. When put into edit mode, the listing should be redisplayed, but this time the selected item should be put into input boxes. If the user clicks another edit link before submitting the current item, the current changes should be discarded.

I'll start by adding another table column containing the link that puts us into edit mode:

```
<TD><A
href="EducationHistory.asp?personID=
<%=listing("personID")%>&
```

listing 1

```
<HTML>
  <HEAD>
    <TITLE>Education History</TITLE>
  </HEAD>
  <BODY>
    <H1>Education History</H1>
    <TABLE width="100%" border="1">
      <THEAD>
        <TR>
          <TH>Year</TH>
          <TH>Institution</TH>
          <TH>Degree</TH>
        </TR>
      </THEAD>
      <TBODY>
        <TR>
          <TD>1990</TD>
          <TD>University of New Orleans</TD>
          <TD>Masters of Science in Computer Science</TD>
        </TR>
      </TBODY>
    </TABLE>
  </BODY>
</HTML>
```

listing 2

```
<%
Dim listing
Set listing = fetchEducationHistory(getPersonID())
Do While Not (listing.EOF)
  %>
  <TR>
    <TD><%=listing("YearNUM")%></TD>
    <TD><%=listing("InstitutionNM")%></TD>
    <TD><%=listing("DegreeNM")%></TD>
  </TR>
  <%
    listing.MoveNext
  Loop
listing.Close
%>
```

listing 3

```
<SCRIPT LANGUAGE="vbscript" RUNAT="Server">
  Public Function fetchEducationHistory(byval personID)
    Set fetchEducationHistory = Server.CreateObject("ADODB.Recordset")
    fetchEducationHistory.Open "SELECT * FROM " & TABLE_NAME & " WHERE
    PersonID = " & personID & "
    ORDER BY YearNUM", CONNECTION_STRING, adOpenForwardOnly, adLockReadOnly
  End Function

  Public Function getPersonId()
    getPersonId = Request("personID")
  End Function
</SCRIPT>
```

```
itemID=<%=listing("itemID")%>">Edit/</A></TD>
```

I'll also need to modify the code that produces the listing so that it includes an editable line. To keep life simple, it's always good to put the editable line at the beginning of the listing—here, I might just insert the code in listing 4 immedi-

ately before the code in listing 2. This almost works, but it's still outputting the read-only row for the selected item. To avoid this, I'll add an optional WHERE condition to the fetchEducationHistory function that excludes the selected item.

The problem now is that both listing 2

a guy walks up to steven spielberg and says,
 "how do I become a movie director?"
 spielberg says, "to be a movie director all you have to do
 is say you are one." the guy says, "that's all i have to do?"
 spielberg says, "yeah. now if you want to be a good movie director . . . "

So far being an XP project has been a lot like being a movie director, but how do we make sure our projects turn out closer to *Schindler's List* than *Plan 9 from Outer Space*? Clearly, there is some difference between saying you are eXtreme and really being eXtreme.

A lot of emphasis is being placed on the twelve practices, but is that the key to knowing if your project is really doing XP?

the twelve practices

XP is most often characterized by 12 practices:

- ✓ Planning Game
- ✓ Small Releases
- ✓ Metaphor
- ✓ Simple Design
- ✓ Testing
- ✓ Continuous Integration
- ✓ Pair Programming
- ✓ Collective Ownership
- ✓ Refactoring
- ✓ 40-Hour Week
- ✓ On-Site Customer
- ✓ Coding Standards

Should we evaluate a project's XPness by its adherence to these practices?

Is a project eXtreme if it follows all 12?

Is it eXtreme if it follows a specified 5 practices and any 4 of the other 7? If it is, what

6

are the 5, and how do we measure adherence?

Can you pick and choose and still be on the path to eXtreme? Would the answers to these questions help us know if a project is XP?

No, the twelve practices are intended to be a starting point. Projects who want to know how to start XP are pointed to the twelve practices and told to do them for a couple of iterations and then to modify and adapt them to their local circumstances.

So, if the directions on the label say you should change how you use the practices, they cannot be the standard against which to decide if a project is eXtreme.

If following the 12 practices isn't the necessary and sufficient condition, how can we tell if the project we are looking at, or working on, should be characterized as an eXtreme Programming project?

first principles

Kent Beck envisioned a new software development process based on 4 values:

- ✓ Communication
- ✓ Feedback
- ✓ Simplicity
- ✓ Courage

That process became eXtreme Programming and they are where we should look when judging a project's status as eXtreme. Communication, Feedback, and Simplicity work together to instill the programmers, customers, and management with Courage. As we see in the figure below, courage is the

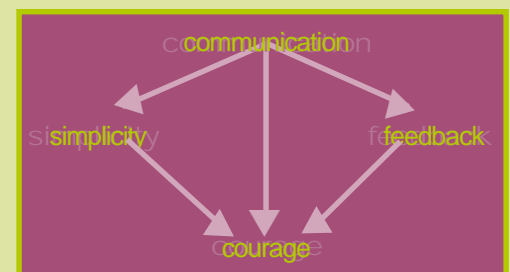
goal of XP. Courage to change the code when it needs to be changed. Courage to ask for new features. And, courage to believe the results of the planning game.

Courage is the trademark of eXtreme Programming. It is our goal. And, it is the yardstick we should use to judge a project.

courage
 (and the lack of it)
 in practice

A team's courage is revealed in how it responds to stress. If every project went according to how it was originally envisioned, the team would not need courage, it would not need XP. XP was designed to allow a team to absorb unexpected changes, to use the new inputs in its next planning session and inform the world of the impact. Non-XP teams quite often react to new inputs by either ignoring them or by believing that the new information will have an insignificant impact on the previously made plan.

It has been my experience that teams that truly follow all 12 practices will have courage. But, I have been in situations when we thought we were doing a good job only to discover that we lacked the required courage to do the job correctly.



when is it not xp.~

by chethendrickson

Usually the lack of courage can be traced to a failure to follow one or more of the 12 practices. That is where we should look to regain our courage, but courage should be our goal and the practices our means for achieving it.

Our goal is to have courage that is based upon an understanding of the state of our project and our teams ability to continue to deliver value. A team that does not have this information can still have courage. But, that courage is built on ignorance. Both Custer and Sitting Bull had courage. Sitting Bull also had repeating rifles and a whole lot more men. His courage was well founded. This is why XP places so much emphasis on communica-

tion and feedback. It is only through knowing where you are and how well you are performing that your courage will not end up being merely foolhardiness.

A team I once worked on was nearing the end of the second phase of an enterprise-altering piece of software. Several months before we had successfully launched the first phase and then immediately started the second phase, which was to increase the scope of the software in several directions. As we approached our projected launch date, the customers began adding new requirements, often canceling them after the programmers had them nearly completed. While each of the new requirements would have improved the

product, in retrospect, I see that the customers were driven more by fear than by a desire to have a more complete program. The first phase had completely changed how a small group within the department did its work. The customers feared that they would not be able to adapt their work force to the new way of working that the new software would require. We should have recognized their fear and addressed it. We should have improved the feedback loop by having more and smaller releases, we most certainly should have introduced their concerns into the planning game. But the outcome was that the second phase was never launched, and even though we were

>> back cover

Chet Hendrickson was at eXtreme Programming's ground zero, the Chrysler Comprehensive Compensation (C3) system. As a developer on the pre-XP C3, he saw how poor communication, inadequate testing, and an overly complex design could doom a development effort. He helped make the decision to throw away 14 months of work and begin again under the guidance of Kent Beck, Martin Fowler, and Ron Jeffries. In a talk at OOPSLA'97 along with Jim Haungs and Rich Garzaniti, Mr. Hendrickson was the first to report on the Chrysler Methodology—the term eXtreme Programming had not yet been coined.

Mr. Hendrickson and Mr. Jeffries spoke on XP at Smalltalk Solutions '97 and '98. Also, in 1998, Mr. Hendrickson carried the XP banner in 'The Project Manager Games' at OOPSLA'98, in Vancouver, BC, and the audience selected him as the manager for

whom they would most like to work. Also at OOPSLA'98, he ran the first birds-of-a-feather session devoted to eXtreme Programming. Here, conference attendees were able to discuss XP with Kent Beck, Ward Cunningham, Ron Jeffries, Rich Garzaniti, as well as with Mr. Hendrickson.

Mr. Hendrickson is an independent consultant, helping software teams improve the software development process by the application of XP's core values of simplicity, communication, feedback, and courage.

Mr. Hendrickson's first book, *Extreme Programming Installed.*, (Addison-Wesley 2000), the second in the eXtreme Programming series, is a collection of connected essays on practices that are presented in the order in which they would actually be implemented during a project.

Mr. Hendrickson can be contacted at richard.hendrickson@acm.org.



7

version control

jgraves@nolacom.com



NOLA developer Joseph Graves is at work on Cyndrus ADS (Asset Decision Support). Cyndrus, Inc. is a software development company formed by NOLA to develop, market, and support a suite of asset management software. Joseph has been with NOLA for over two years, and in his leisure time you may see him playing foosball downstairs at corporate headquarters.

the problem

One of the challenges developers encounter most often has to do with ever increasing sizes and complexities of software development projects. Larger projects require more developers to write more code in more stages. As the scope of software grows, the ability for developers to envision the overall system is compromised.

For one thing, each modification erases previous versions, so it's harder to backtrack if the modifications create new problems. Also, if multiple developers are working on the software—as is necessary on large projects—it's more difficult to coordinate changes, particularly when each developer works on his or her own copy.

early solutions

In 1975, AT&T created a program called Source Code Control System (SCCS) that allowed its many software engineers to trace the history of the software on which they were working. The program was written shortly after the migration from punchcards to mainframes, when disk storage of software became possible.

Amazingly, many contemporary *revision control systems*, as we now call them, still build on the old SCCS approach. SCCS introduced the ideas of version numbering as well as storing all versions in one file and simply logging the differences between versions. It was possible under SCCS, though, for *anyone* to update the revision archive, which paved the way for developer collisions.

The next evolution was the Revision Control System (RCS), an extension of SCCS that added the idea of check-in/check-out. Only one developer could modify a file at a time, because that developer would check out the file and lock it so that only modifications by that developer would be accepted. Other developers had to wait



for the file to be unlocked or checked back in.

In the sixteen years since RCS was created, its influence is still present, as many revision control systems are merely implementations of the same idea. Merant PVCS, Microsoft Visual Source Safe, and others use the check-in/check-out model. Because each check in/check out is logged, these products are exquisitely suited for lock-down development environments where multiple developers work on different files in the same project, and it is extremely important to identify the person who implemented a particular change.

Soon after RCS was developed, another product, Concurrent Versions System, was developed using a different approach. CVS allows for concurrent development on the same file, and does not worry about single files as much as entire structures. If one developer updates the archive and another tries to update the archive with different changes, the second developer will be required to merge the two sets of changes.

current options

Of the many version control options available today, PVCS is one of the most popular and it's used frequently at NOLA. PVCS is based on the RCS model of single file check-in/check-out, but is complemented with extra features and additional packages.

PVCS allows you to define triggers—

actions that should run in response to other actions. These triggers can perform many tasks, including the population of build directories and the execution of database scripts. These tasks can greatly simplify application deployment.

The PVCS suite of applications includes such other helpers as PVCS Tracker, which assists with bug reports and customer service, and Configuration Builder, a tool for the build and deployment process.

Many development systems—such as Java IDE's—are written to integrate with PVCS, but it's not the only option. Microsoft's Visual Studio includes Visual Source Safe, so, when developing in this environment, version control is integrated. CVS is free and open source, so it is frequently integrated into development environments.

One option that is extremely unpopular—not to mention unwise—is no version control system at all. Although this option may seem the most convenient, generally version control applications save a lot of time over the entire project life cycle. All Fortune 100 companies mandate the use of a version control system because of the ability such systems provide for managing the development process.

in summary

Version control and source management has progressed into being one of the core aspects of software development, along with requirements gathering, design, coding and testing. Revision control and source systems have become integral to development environments at large, and to small software development shops, and are even an integral part of open-source distributed development. The one refrain that echoes from all of these developers "We must have a version control system."

"abc"
 "xyz"
 "ghi" "mno"
 "set"



sbaber@nolacom.com

recently graduated from Tulane University, where he earned degrees in computer science and molecular biology. Syed joined the NOLA team in 2000 and is now an eXtreme Programmer in the Commercial Applications Group.

In our last issue, Syed Baber asked, "How many ways can you arrange the alphabet using three different letters in each arrangement? Valid arrangements can be mop, set, abc, or xyz, but not bee, ewe, or cba. Bee and ewe each contain the same letter twice. You could use either abc or cba, but not both, since they use the same letters."

Apparently, there was some competition among the members of the Cyndrus team to find the answers to this puzzle. Five of them submitted correct answers, each with a different solution.

Savita Pradeep wrote that she attempted to solve the puzzle using Java code. She entered the three letter combination string and extracted combinations with duplicate letters, duplicates of previous combinations, and combinations that contained the same letters as previous combinations in a different order. After extracting these duplications, she had 2600 combinations left. She even submitted a list of all the possible combinations she found using the Java code but, since it's twelve pages long, we decided not to print it here.

Joseph Graves and Conrad Carriere both used the mathematical formula for

combinations, which is $C(m,n)=m!/(n!(m-n)!)$, where $m=26$ and $n=3$. They both punched the numbers into the formula and got 2600, right again.

Wardell Bellanger wrote code using Visual Basic, setting up an array of letters. As he looped through the array and found different combinations of the letters, he sent the combinations to a database, where they would only be added if they didn't already exist. He kept a record count of the combinations, and when the array was complete, had 2600 in the database.

Narti Kitiyakara designed a system that would run the string of combinations of three letters at a time and accept combinations that met the requirements for the puzzle. When it was run, it came up with 2600 combinations.

So we have five correct answers and four different ways of solving the problem, varying in complexity. The names of all five contestants who got the right answer were entered in a drawing.

Congratulations to Savita Pradeep whose name was drawn. Savita received a \$50 gift certificate for the restaurant of her choice and invited her Cyndrus comrades to join her for lunch.

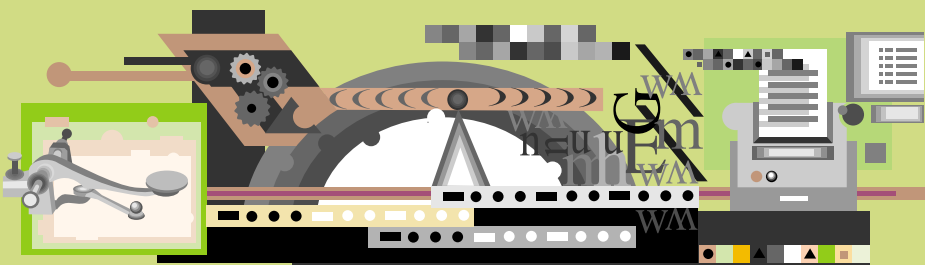
puzzle guy

is a regular feature in peer to peer.

All readers can be eligible for a free lunch worth \$50.

Just fax 504.488.9955 or e-mail sbaber@nolacom.com your correct answer by September 15 and we'll include your name in a drawing that will determine the winner.

And here's the new puzzler:



Since ancient times, cryptic messages have fascinated us. We have attempted to decrypt hieroglyphics and many other lost languages. Many of us can remember magic decoder rings and how we spent

countless hours amusing ourselves with them.

During World War II, Navajo Indians enlisted as Marines used the Navajo language as an encrypted

message. Encryption, which is used to communicate messages meant to be secure and secretive, comes from those roots. This edition's puzzle is to decrypt a message. Here is an example:

♣ ♠ ● ● □ ♣ □ □ ● ♠ ♣ = Hello World!

Keeping in mind the current conditions around the world, what is the meaning of the symbol arrangement below? HINT: Anybody with a word processor can figure this out.

♣ ♠ ♣ ♠ ♣ □ ■ ♣ ♣ ♣ ♣ ♣

and listing 4 output a table row, so I'd have to modify both bits of code if I wanted to reorder the columns or make any other changes. To avoid this, we can extract the common part to its own method,

outputListingRow, then I'll only have to make one change if I want to change the table structure. This method, along with the helper methods for listing 4, is shown in listing 5, which replaces the entire server-side script block in the HTML header. Listing 6 shows the `TBODY` section modified to take advantage of the new method. This is less than ideal, since there's still an external coupling between the code and the HTML, but it's a compromise that I'll have to live with because neither ASP nor straight JSP gives me much choice in the matter.

We're making good progress, but we're still not updating the actual record in the database. To do this, I'll put some code at the top of the `BODY` tag:

```
<%
    If isUpdateSubmission() Then
processUpdateInformation
    End If
%>
```

The code for the two methods used here, shown in listing 7, is fairly simple. The `isUpdateSubmission` function just checks to see if the Submit button has been pressed and that a valid `itemID` has been given. The `processUpdateInformation` subroutine updates the database with the appropriate information. This works fine, but we don't leave edit mode after the update. To do this, I'll add another condition to the `If` statement controlling the editable line:

```
If getSelectedItem() <> NO_ITEM And
Not (isSuccessfulUpdate()) Then
```

I'll also need to modify the `getSelectedItemCondition` function so that it doesn't put a restriction on the listing following a successful update. (The code for these changes can be seen in listing 9, which you can view at www.nolacom.com.

And that pretty much settles the update portion

listing 4

```
<%
    If getSelectedItem() <> NO_ITEM Then
        dim institution
        dim degree
        dim year

        retrieveHistoryItem getSelectedItem(), institution, degree, year
    %>
    <FORM
action="EducationHistory.asp?personID=<%=getPersonId()%>&itemID=<%=getSelectedItem()%>"
method="post">
        <TR>
            <TD><INPUT type="text" name="year"
value="<%=year%>" /></TD>
            <TD><INPUT type="text" name="institution"
value="<%=institution%>" /></TD>
            <TD><INPUT type="text" name="degree"
value="<%=degree%>" /></TD>
            <TD><INPUT type="Submit" name="Submit"
value="Submit" /></TD>
        </TR>
    </FORM>
End If
```

listing 5

```
<SCRIPT LANGUAGE="vbscript" RUNAT="Server">
    Public Function fetchEducationHistory(byval personID)
        Set fetchEducationHistory = Server.CreateObject("ADODB.Recordset")
        fetchEducationHistory.Open "SELECT * FROM " & TABLE_NAME &
            " WHERE " & PERSON_COLUMN & " = " &
personID & getSelectedItemCondition() & " ORDER BY " & YEAR_COLUMN,
CONNECTION_STRING, adOpenForwardOnly, adLockReadOnly
    End Function

    Public Sub outputListingRow(byval institutionCell, byval degreeCell,
byval yearCell, byval editCell)
        Response.Write "<TR>" & vbCrLf
        Response.Write " <TD>" & yearCell & "</TD>" & vbCrLf
        Response.Write " <TD>" & institutionCell & "</TD>" & vbCrLf
        Response.Write " <TD>" & degreeCell & "</TD>" & vbCrLf
        Response.Write " <TD>" & editCell & "</TD>" & vbCrLf
        Response.Write "</TR>" & vbCrLf
    End Sub

    Public Sub retrieveHistoryItem(byval itemID, byref institution,
byref degree, byref year)
        Dim cursor
        Set cursor = Server.CreateObject("ADODB.Recordset")
        cursor.Open "SELECT * FROM " & TABLE_NAME & " WHERE " & ITEM_COLUMN &
            " = " & itemID, CONNECTION_STRING, adOpenForwardOnly, adLockReadOnly
        institution = cursor(INSTITUTION_COLUMN)
        degree = cursor(DEGREE_COLUMN)
        year = cursor(YEAR_COLUMN)
        cursor.close
    End Sub
</SCRIPT>
```

of our code. Of course I've ignored such issues as error handling and security, but this is just a prototype and our customer doesn't feel that these issues are important at present.

Now that we've finished the edit capabilities of our system, let's turn our attention to the process for adding a new item. In terms of both the presentation and the coding, the add capability is pretty similar to the edit capability. Visually, I'll just add a link to the bottom of the listing that puts us into an "add item" mode:

```
<TD colspan="4" align="center">
    <A
href="EducationHistory.asp?mode=
add&personID=
<%=getPersonId()%>">New Item</A>
</TD>
```

Next, we have to modify the existing

code so that it will handle an "add" case—specifically, I'll modify the conditional that controls whether or not the input boxes are shown. (And since I've changed it so much, I've taken the opportunity to move its logic into a method called `shouldDisplayInputBoxes`, see listing 8.)

We'd also need to modify `retrieveHistoryItem` so that it returns empty strings when no item is selected and `processUpdateInformation` so that it handles an add case. (This makes a name change worthwhile as well; the new name will be `processSubmission`.) All of these changes are fairly simple, so I won't show them until listing 9, which can be viewed at www.nolacom.com.

The final story we need to implement is

the delete function. To do this, I'll add one more column to the table. This will be a link, just like the edit link, except that I'll add the "mode" keyword *delete*. I'll also add methods analogous to `isInformationSubmitted` and `processSubmission`, which will check for the *delete* mode and do the work of the deletion. I also need to modify the `shouldDisplayInputBoxes` function so that it returns false when we have a deletion. The final code is shown in listing 9.

Before we draw any conclusions, let's consider how a JSP version would differ from the ASP version. Apart from the obvious—code written in Java instead of VB—there would be no difference. The structure of both pages would be exactly the same. Both technologies almost require you to mix your business logic with your presentation logic. Even if we moved the script block into COM or Java objects, we'd still need the logic controlling the number of rows displayed and whether or not to display the input boxes. This is unfortunate, since it makes it harder for an HTML person to make modifications to the page. My experience with this has been: I write an ugly but functional page like this and ask an HTML person to make it pretty. They just look at the output and give me back another page hard-coded with a few sample rows. I then have to modify my original page to make it look like what they wanted. My ideal would be to make a prototype page like this, give it to the HTML person, and let them make the modifications directly.

In the next article, we'll look at a couple of technologies (ISAPI and Java custom tags) that might make this more feasible by doing a much better job of decoupling the presentation from the business logic. </end>



nkityakara@nolacom.com

taught himself nearly a dozen programming languages before earning a b.s. and an m.s. in computer science from the american institute of computer sciences. narti joined the nola team in 1998 and is now a lead systems architect in the commercial applications group.

listing 6

```
<TBODY>
  <%
    If getSelectedItem() <> NO_ITEM Then
      dim institution
      dim degree
      dim year

      retrieveHistoryItem getSelectedItem(), institution, degree, year
    %>
    <FORM action="EducationHistory.asp?personID=<%=getPersonID()%>&itemID=
    <%=getSelectedItem()%>" method="post">
      <%outputListingRow _
        "<INPUT type='text' name='institution' value='" &
        institution & "' />", _
        "<INPUT type='text' name='degree' value='" & degree & "' />",
        "<INPUT type='text' name='year' value='" & year & "' />", _
        "<INPUT type='Submit' name='Submit' value='Submit' />"%>
      </FORM>
    <%
  End If
  %>
  ^%>
  Dim listing
  Set listing = fetchEducationHistory(getPersonID())
  Do While Not (listing.EOF)
    outputListingRow _
      listing(INSTITUTION_COLUMN), _
      listing(DEGREE_COLUMN), _
      listing(YEAR_COLUMN), _
      "<A href='EducationHistory.asp?personID=" & listing("personID") & "&itemID="
      & listing("itemID") & "'>Edit</A>"
    listing.MoveNext
  Loop
  listing.Close
  %>
</TBODY>
```

listing 7

```
Public Function isUpdateSubmission()
  isUpdateSubmission = Request("Submit") <> "" And getSelectedItem() <> NO_ITEM
End Function

Public Sub processUpdateInformation()
  Dim cursor
  SuccessfulUpdate = false
  Set cursor = Server.CreateObject("ADODB.Recordset")
  cursor.Open "SELECT * FROM " & TABLE_NAME & " WHERE " & ITEM_COLUMN &
  " = " & getSelectedItem(), CONNECTION_STRING, adOpenDynamic, adLockOptimistic
  cursor(INSTITUTION_COLUMN) = Request("institution")
  cursor(DEGREE_COLUMN) = Request("degree")
  cursor(YEAR_COLUMN) = Request("year")
  cursor.Update
  cursor.Close
  successfulUpdate = true
End Sub
```

listing 8

```
Public Function shouldDisplayInputBoxes()
  shouldDisplayInputBoxes = (getSelectedItem() <> NO_ITEM
  Or isAddCommand()) And Not (isSuccessfulUpdate())
End Function

Private Function isAddCommand()
  isAddCommand = getCommandMode() = "add"
End Function
```

delivering new features every three weeks writing one hundred new unit tests a month, pair programming every day, and following every other practice, we were not XP.

fear is the mind killer

The lack of courage — what I like to call “fear” — is self-perpetuating and can most often be traced to an initial lack of one or more of the other three values. A group of programmers who fear doing the simplest thing that could possibly work because they do not have adequate acceptance testing (a lack of feedback) may later lack the courage to optimize performance due to the system's complexity. Fear will prevent you from doing the right thing, it reduces your degree of freedom and will eventually drag your project to a halt. This is the antithesis of eXtreme Programming.

conclusion

Delineating the twelve practices has made it much easier for teams to start doing XP on their own. We should

remember that the practices are merely the suggested route and that we need to focus on the values behind XP. Simplicity, communication, feedback, and, most important, courage. Over time we will find new practices and new combinations of the original practices that make sense in a particular situation. The practices will change, the values will not.

Even though we are doing most or all of the practices, we need to pay attention to our courage level to know if we are an eXtreme project. How do we react when it is suggested to change from monthly iterations to two weeks? Do we approach a performance problem with a patchwork of solutions or do we find the best solution and implement it everywhere it is needed? Do we address the fear in ourselves and our customers that may stop us from doing the right thing? That is what an XP team would do. </end>

not
xp?



PRSRT STD
US POSTAGE
PAID
NEW ORLEANS LA
PERMIT NO. 2554