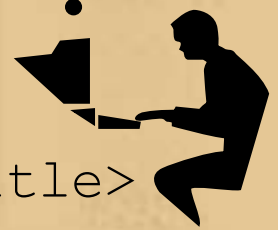




<title>peer to peer</title>



The First Computers

from the Difference Engine to the Mark I

1833

1930

1939

1940

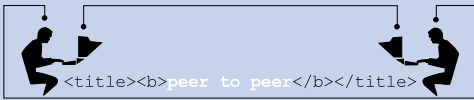
1944



also inside



AVIGNON
Acceptance Testing HTML
by Narti Kitiyakara



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 about the field of information technology.

We offer a broad range of IT services to the public and private sectors. Our services include information management consulting, Internet/intranet/extranet development, system architecture, and project management.

For inquiries, please call us toll free at 888.488.1101 or visit our Web site at www.nolacom.com.

peer to peer thanks

Horst Zuse for his kind permission to reprint the photograph of his father's Z1, cover and page 5.

The XP/Agile Universe 2002 and NCS Proceedings for kind permission to reprint "Acceptance Testing HTML," beginning on page 4.

The Grace Hopper Collection, Archives Center, National Museum of American History, Smithsonian Institution, for kind permission to reprint the photograph of Ms. Hopper on page 3.

The Charles Babbage Institute for kind permission to reprint the photograph of the Difference Engine on page 5.

Iowa State University, Ames Laboratory, for kind permission to reprint the photograph of the ABC on page 5.

The Naval Historical Foundation for kind permission to reprint the photographs of Grace Hopper and the first computer bug on page 3 and the back cover.

Trish Thomas edits and manages the publication of peer to peer. Bonnie Bart designed this issue. Syed Baber, Danielle DeAses, Narti Kitiyakara, and Andrea Washington contributed. We welcome your comments at peertopeer@nolacom.com.



iso 9001 registered

Atlanta
Houston
New Orleans
Washington D.C.

Corporate Headquarters
3535 Canal Street
New Orleans, Louisiana 70119
Voice: 504.488.1111
888.488.1101
Facsimile: 504.488.9955
www.nolacom.com

in this issue...

Beginning with this issue, look for a series of articles about computer and Internet history. Get a refresher in computer history on page 5. What you didn't know may surprise you! *new!*

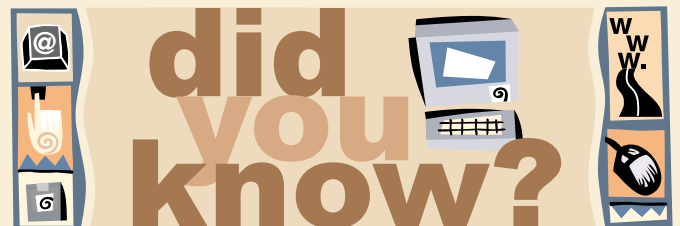
Narti Kitiyakara discusses problems associated with acceptance testing. Find out how the commercial applications group developed a solution, page 4.

Did you know? This new feature contains IT facts you may not know. Look below to see if you knew. *new!*

She claimed it was laziness. Whatever the reason, Grace Murray Hopper forever changed the way computers were used and programmed. Read about her achievements (and find out which NOLA team member knew her) in an article by Danielle DeAses, page 3.

See if you can solve the new puzzler from Syed Baber on page 9! Remember, if you send in the correct answer by the deadline you'll be eligible for the free-lunch drawing.

Share your brilliance! We welcome contributions from all NOLA team members. Why write for peer to peer? Besides the opportunity to share your ideas, humor, or expertise, there is an added bonus: we offer honorariums to contributing writers. Contact Trish Thomas for guidelines at pthomas@nolacom.com. €



The name of the world-famous search engine, Google, is a play on the word googol. "Googol" was coined by Milton Sirota, nephew of American mathematician Edward Kasner, to refer to a number represented by one followed by one hundred zeros. The mission of Google's creators was to organize the vast amount of information on the Web, so a name that refers to such a large number seems appropriate.

Creators Sergey Brin and Larry Page were Ph.D. candidates in computer science who were fascinated by meaningful patterns in Web link structure, especially backlinks, which are pages linked back to a site. When they presented their idea to the first investor, the investor wrote a check to Google Inc. After a few weeks, Brin and Page decided it might be a good idea to open an account in the name of Google Inc. so that they could cash the check. And the rest, as they say, is history...

dare&do



ddeases@nolacom.com

newly married danielle deases is a psychology student at the university of dallas. no time for hobbies, she spends all her spare time studying to complete her degree. that is, except when she's dreaming up ways to decorate her new home.

Grace Murray, born in New York City on December 9, 1906, became one of the most significant people in the history of modern computers. Her motto, "Dare and Do," reflected her philosophy that if you have an idea for something that will work

In 1943, because of World War II, she decided to join the Navy WAVES (Women Accepted for Voluntary Emergency Service). Of course this was no easy task. At 34 years old and 105 pounds, she was, by Navy standards, both overage and

Reserve. She trained at Midshipman's School for Women and graduated first in her class.

Lieutenant JG Hopper was assigned to the Bureau of Ordinance Computation Project at Harvard University, under Professor and Naval Reserve Lieutenant Howard Aiken. She worked with the rest of the small research team on the Mark I, the world's first large-scale, automatically sequenced, digital computer, used to calculate aiming angles for Naval guns. Since this task was so important to the war effort, the team often worked through the night. Thus she was initiated into the world of information technology.

In 1946 she turned down a renewal with Vassar to continue her research at Harvard, where she worked on the Mark I's successors, the Mark II and Mark III. She was credited with coining the term *bug* after tracing an error in the Mark II to a moth that had become trapped in the machine. The bug was

carefully removed and taped in the logbook. She later used the term *bug* to refer to programming errors as well as hardware problems.

She didn't remain at Harvard long. That same year she left to join the Eckert-Machly Computer Corporation, later called Sperry Rand. At Eckert-Machly she helped design the UNIVAC, the first commercial electronic computer, which operated a thousand times faster than the Mark I.



Lt. Hopper at her desk in the Computation Lab, 1947.

She was credited with coining the term *bug* after tracing an error in the Mark II to a moth that had become trapped in the machine.

better, faster, and easier—then do it. She repeated the adage, "It is easier to ask forgiveness than permission." By living boldly, she advanced technology in ways that made modern computers possible.

"Amazing Grace" graduated Phi Beta Kappa with a B.A. in mathematics from Vassar in 1928. She earned an M.A. from Yale in 1930, the same year she married Vincent Foster Hopper. She returned to Vassar in 1931 to teach mathematics for a starting annual salary of \$800. In 1934, she earned a Ph.D. from Yale. By 1941, she was an associate professor and had won a faculty fellowship at NYU's Courant Institute for Mathematics.

underweight for enlistment. But these obstacles didn't slow her down. She obtained special permission from the government and a leave of absence from Vassar, and was sworn into the U.S. Naval

underweight for enlistment. But these



The first computer bug. In 1945, a moth was removed from Relay #70, Panel F, of the Harvard University Mark II Aiken Relay Calculator.

>> back cover

Acceptance Testing HTML

We have been conducting an XP project for nearly two years. During this time, we have experimented with many tools and techniques for acceptance testing. Here we discuss the relative costs and benefits that we've found using a commercial testing tool, manually executed tests, and hand-coded Java tests.

Avignon, an XML-based extensible scripting language that we developed, allows us to specify acceptance tests at a high level, in advance, and with relative ease.

Project Background

In late 2000, with two developers and a project manager-customer, we embarked on our first XP project. Our goal was to develop a Web-based J2EE application for commercial release. Over the next eighteen months, the project grew to six developers and consisted of over 650 Java classes, 80 Java Server Pages, and 35 database tables. Our initial process was based on *Extreme Programming Explained*¹ and the developers tried to follow all of the programmer practices, including pair-programming, unit-testing, and iteration planning. Automated acceptance testing, however, was to prove a troublesome issue.

Tools for Testing HTML

Manual Tests

When we first began the project, the customer prepared "stories" of up to thirty typed pages along with detailed acceptance tests. The plan was that the developers would use the tests to get detailed information about the story, but the customer would manually execute the test to determine whether or not to approve the story. The Quality Assurance Department was also supposed to run the previous acceptance tests on a regular basis to make sure that adding the new story had broken no previous stories.

The developers dutifully ran acceptance tests by hand for each

nkityakara@nolacom.com

narti kityakara 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.



"Automated Testing HTML" first appeared in Extreme Programming and Agile Methods, a publication of the XP and Agile Universe Conference, held in Chicago in August 2002.

completed story, but it turned out that the tests were so detailed that it was easy to fail on trivial points that were not noticed. It also turned out that some aspects of the manual acceptance tests were subject to differing interpretations. Thus the developers would legitimately feel that they'd passed an acceptance test, but the customer would say that they had not. Apart from that, neither the developers nor the customer nor the QA Department was running the previous tests on a regular basis. It was only by luck, therefore, that the development team would discover that modifying existing code had changed the functionality of the application. This state of affairs led us to purchase a full-fledged acceptance-testing package.

Commercial Testing Tools

In order to try to alleviate the problems of manual acceptance testing, we evaluated several commercial testing packages, including Segue Software's SilkTest, Empirix's eTester, and Compuware's QARun. All of the packages were rated on the basis of applicability to projects we were undertaking, ease of use, ability to test multiple server and client environments, and cost. As a result of this evaluation we purchased Compuware's QARun.²

Shortly after the evaluation of automated testing tools, two of us attended ObjectMentor's XP Immersion class. It was there that Ron Jeffries impressed upon us the need for automated acceptance tests that the developers could run themselves.³ When we returned from the Immersion, we set about trying to convince our customer of the importance of getting the acceptance tests automated and giving the developers the ability to run them. Unfortunately, it turned out that the customer would find that the scripting facility in QARun was not amenable to writing the acceptance tests in advance.

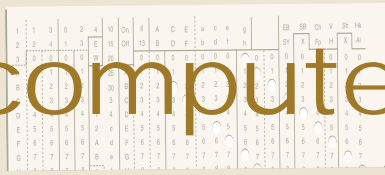
HTTPUnit

The customer had never disputed that the acceptance tests should be automated, but still felt that they should be done after the fact by the QA Department. At this point, however, QA was still not able to run the previous tests in a timely enough manner to warn the developers of impending problems. So the developers, on their own initiative, started using HTTPUnit⁴ to manually code the acceptance tests. They had discovered the program perusing the XP Web sites. This proved quite effective at stopping us from changing old code, but interpreting the customer's ideas of how to test the details in the visual components was difficult.

>> page 6

the first computers

by danielle deases



no one would deny that our lives would be completely different today without the invention of the computer. I would have written this article by hand, for instance, and then have had it typeset. Well, I wouldn't even be writing this particular article in the first place. Most of us can't imagine life without computers; I can't even imagine life before Windows. Yet, as important as the invention of the

perhaps the largest computational device in the world in its time, weighing approximately 100 tons. The Rockefeller Foundation funded the construction of this analog device, which could solve simple equations and measure movements and distances sort of like an automatic slide rule.

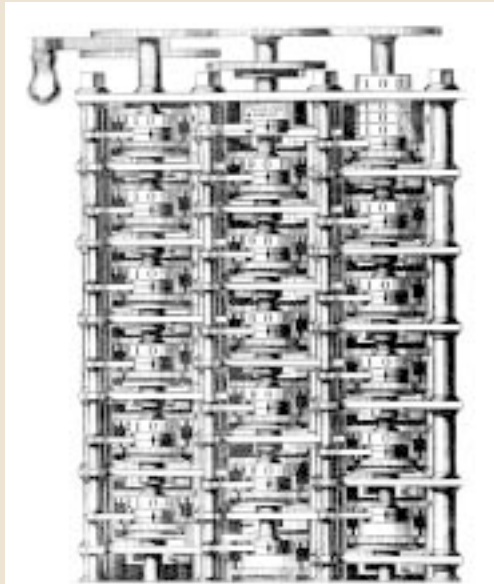
Possibly because its function was digital rather than analog and it used the binary system, the Z-1 is sometimes referred to as the world's first computer. It was built by German engineer Konrad Zuse in his parents' living room in 1935. Zuse continued to improve his design in the Z-2, Z-3, and Z-4. The Z-3 was the first fully functional program-controlled electromechanical digital computer, and was completed in 1941.

Unfortunately, all but the Z-4 were destroyed during World War II, and the original designs were not discovered until well after the end of the war.

Since no one knew about Zuse's machines for many years, John Vincent Atanasoff and John Berry at Iowa State University were able to take credit for constructing the "first computer" in 1936. The Atanasoff-Berry Computer, later dubbed the ABC, was developed for the computation of linear equations in physics. It used components that became elements of later, more advanced

computers, such as the electronic arithmetic unit and regenerative memory.

Of course, the ABC was a special-purpose machine, so those looking to the origination of the general-purpose computer wouldn't stop at Atanasoff. Using the general-purpose criteria, the last two contestants in the "Who Really Invented the First Computer?" game come to mind. The first of the two is George Stibitz, a researcher at Bell Labs, who designed and built the Bell Labs Model Relay Computer in 1940. Stibitz's



Charles Babbage's Difference Engine No. 1, 1853, led to development of the Analytical Machine.

computer was, there is some disagreement as to who is responsible and exactly when it all started.

Some say that computer history started as early as 1833 with Charles Babbage's Analytical Machine. Designed for the calculation of mathematical tables, his machine would have contained all of the most basic components of later computers and followed instructions from punched cards. Unfortunately for Babbage, what could have been the world's first general purpose computer was never completed. Called the "Father of the Computer," he never actually built a working computer.

The next contender is Vannevar Bush, who designed and built the Differential Analyzer in 1930. It was



Konrad Zuse's Z-1 set up in his parents' living room.

machine vies for the title of the world's first electronic digital computer. And last, but arguably one of the most famous, is Howard Aiken, who developed the Harvard Mark I— known officially as the IBM Automatic Sequence Controlled Calculator (ASCC)—between 1939 and 1944. Standing at eight feet tall, fifty feet long, and weighing about five tons, the Mark I was the first large-scale (I think "large" is an understatement) automatic digital computer. Its original and most important use was for the calculation of aiming angles for Naval guns during WWII.

So we have six contestants in our game. I'll leave it to you to decide the winner. Of course, who really started it all isn't as important as the fact that these people and many others made possible one of the most important technological developments of the twentieth century. ↵



The Atanasoff-Berry computer, ABC.

Our customer has always been very concerned with the appearance of the application. HTTPUnit provides an API for examining the HTML output of an application, but we found it awkward to use for the level of detail that our customer wanted. It turned out, however, that HTTPUnit would also expose the generated HTML as an XML document, so we could use XPath⁵ to make very fine-grained tests of the HTML without a great deal of coding effort. Unfortunately, as the tests were being developed simultaneously with the code, they didn't help us maintain consistent HTML coding for a given visual effect. They were also quite brittle because the format of the HTML was compiled into the tests. Every change to the appearance of the application required laborious changes to the acceptance tests as well.

Ironically, the very success of the acceptance tests to prevent unintentional functionality changes helped hide their value from the customer. The most visible thing to the customer was the amount of time that it took to code the tests in Java. At that point, though, the developers were still the only group actually automating the test, so the customer couldn't object too strongly. But there was still a feeling that the developers should not be taking up valuable coding time with acceptance tests. For that matter, the developers tended to view the time they were spending on testing as a necessary evil. No one wanted to do away with automated tests that we could run, but they were tedious to write and there tended to be much debate with the customer about what should be tested.

What Can Be Tested?

Everything

At first, after we discovered the value of automated acceptance tests, nothing seemed too trivial to test. Not only were the correct results supposed to be put into the database, but also almost every visual element was checked. Not only were the correct results to be displayed, but every formatting element had to be correct as well. This proved to be very time consuming for the developers and also quite brittle. The acceptance tests generally needed a specific state to be set up in the application before they could be run. This made the developers generate large amounts of code to initialize the state of the application. The customer was also finding that even a small change in appearance required a fair amount of rework in the tests. Switching to a snapshot of the expected HTML would have alleviated this problem, but would have also put us back in the position of needing the application ready before the acceptance test could be done. The snapshot method would have also led to more brittleness in the tests, since the coded version could always choose how to interpret a visual element (i.e., ignoring non-visual elements of the HTML tags, such as `id` attributes, unless they were important in some way).

Functionality Only

When the cost of testing everything became prohibitive, we scaled back testing to only what was necessary to the functioning of the application. Thus, on the HTML side, we tested only things like the `name` and `value` attributes of `input` tags, references for the

`anchor` tags, and basic text of the output. This took some of the burden off the developers, but did not relieve them of the problem of initializing the application state before each test. It also did nothing to solve a problem discovered by the customer even when “everything” had been tested: visual inconsistency between the different HTML pages.

Visual Consistency

Our six developers could pair in fifteen different ways and each of these possibilities could render the same idea in a different way. Although the developers wanted to maintain visual consistency for the customer, this was difficult to achieve. The customer, rightly, did not want to specify too much detail about which HTML to use to achieve a certain visual effect, but different pairs ended up using different HTML to do so and the results were not always the same. One pair might put a page title into an `H1` tag, while another might put it into a `P` tag with the same font size as the `H1`. This broke down when the customer wanted to define a default CSS style for page titles. The development team had to go back and standardize, after the fact, a common way for doing titles. Detailing the visual elements in an acceptance test before the story was started would have helped matters, but no one wanted to go back to the massive coding involved in testing everything. One proposed solution was to write a separate set of tests that checked the visual appearance of the application without testing functionality beyond page generation. This would help the visual consistency issue, since it focused on one visual element at a time, but it would still be brittle and require a large amount of coding. Before embarking on this testing method, we decided to try another approach.

Origins

One of the proposed solutions for dealing with the general problem of visual consistency was to allow the developers to generate XML that would be transformed into HTML by a standard XSL.⁶ This would have removed the problem of deciding how to generate the HTML on a page-by-page basis, and allowed the developers to concentrate on the higher-level concepts of what should appear on the page. Unfortunately, the customer was concerned that XML/XSL was not, at the time, supported by enough platforms to be portable to a large number of environments. Later, however, one of our developers realized that they could still use this system in the acceptance-testing environment to actually run the tests. It could also remove the burden of coding the acceptance tests from the programmers by allowing the customer to make detailed, yet easy to change, specifications of what was expected.

Avignon is Born

Thus came about Avignon, a combination of an XML-based scripting language and a high-level page description language for executing acceptance tests against an HTML application. The customer also helps produce an XSL that converts the page descriptions into HTML. This ensures that the developers generate the same HTML to produce the same high-level concept. Avignon is a living language that is expanded, in consultation with the customer, whenever necessary.

Although the page comparisons performed by Avignon return

to infinity and beyond

by andrea washington

awashington@nolacom.com



andrea washington joined the nola team as an administrative assistant in 1999. when we introduced her in the spring '99 issue of *peer to peer*, we quoted her as saying, "i've worked in a lot of different office environments and feel as though nola is a perfect fit. i see my position here as an endless opportunity to broaden my horizons." three years later, having been promoted three times—first to education coordinator, then to corporate systems specialist—ms. washington is now our quality system manager.

As you may know, the ISO standard under which we are registered, ISO 9001:1994, is being phased out and replaced with ISO 9001: 2000. Along with all other ISO 9001 registered companies, we are required to transition to ISO 9001:2000 by December 2003.

The 2000 standard sets a new pace. It is a process-based approach to quality management that focuses on close monitoring of customer perception, enhanced customer satisfaction, analysis of collected data, and improved communication.

The new standard introduces five major key processes—quality management; management responsibility; resource management; product realization; and measurement, analysis, and improvement—along with corresponding requirements for obtaining and sustaining ISO conformity.

To meet the new *quality management* requirements, we will identify existing

processes and implement new processes. We will also maintain documentation to ensure effective process planning, operation, and control. **Management responsibility** requires the executive management team to demonstrate a commitment to enhanced customer satisfaction by ensuring that customer requirements are understood and met; that the quality policy is implemented and maintained; that quality planning occurs consistently; that responsibility and authority are communicated; and that opportunities for improvement are assessed at planned intervals.

Resource management mandates that we provide and maintain adequate infrastructures, work environments, and resources, and that we sustain competence, awareness, and training of resources. **Product realization** requires that we develop, implement, and maintain customer-related processes in the areas of design and development, purchasing,

production, and service provision.

I think of the final key process—**measurement, analysis, and improvement**—as the checks and balances requirement. It requires that we monitor and measure our processes, analyze the data we gather, and continually improve our daily business practices.

As a team, we all play a role in the functionality of our organization. Continual improvement of our business processes demands the commitment and dedication of our entire organization. To help us achieve a successful transition to the new standard, you can use the tools we have developed to promote continual improvement: our corrective and preventive action process, our employee concerns procedure, and our suggestion box on teamNOLA.com. We want to hear what you think. As Buzz Lightyear said, "To infinity and beyond." There are no true limits except the ones we place upon ourselves. ←

to the brittleness of testing everything, it has proved easy enough to change the expected HTML for every page in the application by changing the XSL that generates it for the test. Because pages are described at a high level, it has also become much easier to pregenerate the expected results. The original page description language was designed for easy transformation into HTML. It quickly became apparent, however, that an even more abstract description could be given on a per-page basis. The abstract description consists only of the elements that can vary on the page. It is then transformed into the original page description language, which is itself transformed into HTML. This makes it extremely easy to change the expected results for any individual page or for every page simultaneously. (An interesting side effect has been that the customer can now quickly prototype the HTML pages entirely outside of the application. This allows the customer to quickly determine the desired look-and-feel.)

Implementation

Implementing Avignon is quite simple. A JUnit test finds all of the files in a directory with the pattern "`*TestSuite.xml`." It passes each of these files through an SAX-based XML parser that validates the syntax of the file and fires an event for the start and end of each XML element. Each element must have an associated Java class matching `ElementNameHandler` class that implements the `AvignonTagHandler` interface. This interface defines two methods:

```
void start(AvignonTestState state,
Attributes attribs)
and
void end(AvignonTestState state)
```

The `AvignonTestState` class allows each tag to access its

>> page 8

parent tags, add messages to the error log and request Web pages from the actual application.

Scripting Language

At this time there is no formal schema or DTD for Avignon, but a brief overview follows.

Test Definition Elements

These elements allow the user to integrate the acceptance tests with our ISO 9001 framework and to initialize the application state before a test.

TestSuite

This is the top-level element for any Avignon test suite file. It has one attribute, `unitName`, which defines the name of the unit being tested. We use this name to help track test results. When this tag ends, it records all of the test results in a database in accordance with our ISO 9001 policies. This element contains any number of `AcceptanceTest` elements.

AcceptanceTest

This element defines an actual test to be performed. It also has one attribute, `testName`, which it gives to its parent `TestSuite` element along with a pass/fail test result. This tag is also responsible for restoring any application state that has been modified by the test. This element may contain one `DatabaseState` element and any number of other test command elements.

TestScript and ScriptParam

These elements allow the user to execute a separate file containing an Avignon script. The `TestScript` element takes a name attribute that tells it what file to execute. Before executing the script, however, it will do a textual substitution of the information provided by the `ScriptParam` elements contained within. This allows the user to execute the same script with different data without having to actually duplicate the script.

DatabaseState & DatabaseTable

These elements allow you to perform low-level database operations before a test is run. This requires the customer to have knowledge of both SQL and the application database, but it is easy enough to add application-specific tags, implemented either with an XSLT that converts them into the low-level tags or with Java handlers that simplify the initialization of the database for the customer.

The `DatabaseState` is used simply to group `DatabaseTable` elements. Each `DatabaseTable` element is responsible for saving the state of the table specified in its attribute. Each `DatabaseTable` element may also have zero or more of the elements below in any combination.

DatabaseInsert

This element allows you to define a SQL statement of the form

```
INSERT INTO table_name [(columns)]
VALUES(values). The table name is defined by the
surrounding DatabaseTable element, columns is defined by an
optional attribute to DatabaseInsert tag. Finally, values is
defined by a required attribute to the DatabaseInsert tag. You
format both columns and values in such a way that the SQL
statement will work correctly when it is executed.
```

DatabaseUpdate

This element allows you to define a SQL statement of the form `UPDATE table_name SET field=value[,field=value]* [WHERE condition]`. The value of `condition` is defined by an optional attribute to this tag, while the set clauses are defined by one or more sub-elements. These sub-elements, `UpdateField`, each take two parameters: a field name and the value to set it to.

DatabaseDelete

This element allows you to define a SQL statement of the form `DELETE FROM table_name [WHERE condition]`. Again the condition comes from an optional attribute to this tag.

Test Command Elements

These elements allow you to perform operations through the application's Web interface. Each of these elements take an optional attribute, `pageDescription`, that allows you to specify an XML description of what the resulting page should look like. The comparison is done by generating the HTML for the actual page and the expected HTML (transformed from

the XML description) and comparing them, without regard to white space. The comparison is done without regard to white space because it proved too difficult to generate the same HTML, including non-significant white space, as was expected. Given that the possibility of incorrectly generating significant white space was small, we chose to ignore white space altogether. Each of these elements also has an optional `preTranslation` attribute. If this attribute is present, the XML specified by `pageDescription` is first transformed by the style sheet defined in `preTranslation` and then transformed into HTML.

MenuClick and PageClick

These two elements are essentially the same, but the former operates on the application's HTML menu while the latter operates on the content frame of the application. Both use `HTTPUnit` to click a link on the page given its text.

SubmitPage

This tag allows you to fill out an HTML form on the current content page and submit it back to the application. It takes the value of the

Avignon is a living language that is expanded, in consultation with the customer, whenever necessary.

Congratulations to Lynne Cantrelle!

She solved the last puzzle correctly and was lucky enough to have her name drawn from a total of 11 correct responses. A member of the human resources and accounting group at corporate headquarters in New Orleans, Ms. Cantrelle will receive a \$50 gift certificate for lunch at her chosen restaurant, Outback Steakhouse.



Peace on Earth!—the correct answer—was also submitted by Michael Ashmead, Phil Bernard, Brady Furr, Joseph Graves, Geri Grubs, David Olivier, Savita Pradeep, Brandy Romain, Garrett Siegel, and Bruce Woods.

And here's the new puzzler:

What three historical computers can you descramble from these letters and numbers?

AACCEIINNRSTUV08



sbaber@nolacom.com

syed baber 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.

puzzle guy is a regular feature in peer to peer.



puzzle guy

All readers can be eligible for a free lunch worth \$50. You don't have to be a nola team member.

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

submit button as an attribute and fills out the input values in tab order (unless otherwise specified) with values given by its sub-elements, `InputValues`. Each `InputValue` element must have a `value` attribute and may have either a `name` or a `position` attribute. If no `name` or `position` is specified, the value goes into the current input box and the focus is moved to the next input box. If a name or position is specified, the focus is first moved to the correct input box, the value is put into that box and the focus moves to the next input box.

DatabaseAssertion

This element allows you to specify that the application's database must be in a given state for the test to pass. You specify three attributes, a `tableExpression`, an optional `whereCondition` and an `expectedCount`. The system generates a SQL statement of the form: `SELECT COUNT(*) FROM tableExpression [WHERE whereCondition]`. The assertion passes if the result of this statement matches the expected count.

User-Defined Elements

The preceding elements represent the code of the Avignon language, but it was never meant to be static. Customers are encouraged to create their own elements. This can lead to some interesting element names (an *EggClick* element, for example), but has proved reasonably successful when the customer consults with the developers about how to phrase what he wants to do, such as whether attributes or sub-elements would be easier. It has also helped to reduce the developers' misunderstandings of what the acceptance tests were doing. Instead of having to describe the whole test to the

developers, the customer now needs to concentrate only on the new elements in the test.

For each new element the customer defines, the developers must add a Java class implementing the `AvignonTagHandler` interface. This class implements the customer's intentions for the tag, be it an assertion or some action command. Because the handlers are dynamically loaded, no recompiling or relinking of the Avignon system is necessary when adding new elements.

An Avignon Example

The following test scripts sets up the test information and calls the "CreateScript.xml" script:

```
<?xml version="1.0" encoding="UTF-8"?>
<TestSuite name="Composer Information">
  <AcceptanceTest name="Add New Composer">
    <TestScript name="CreateScript.xml">
      <ScriptParam
name="ComposerListing"
value="EmptyComposerListingPage.xml" />
      <ScriptParam
name="DatabaseScript"
value="NoComposerSetup.xml" />
      <ScriptParam name="DisplayName"
value="TestDisplay1" />
      <ScriptParam name="ActualName"
value="TestActual1" />
    </TestScript>
  </AcceptanceTest>
</TestSuite>
```

>> page 10

world widewhat?

by danielle deases

Judge Michael Kistrup, Danish provincial court in Copenhagen, has ruled that deep linking is illegal.

Newsbooster.com—a second-generation search engine that specializes in news searches—was found to be in violation of both the Danish Copyright Act and the European Commission's Database Directive on the Legal Protection of Databases, a decision that could pose a threat to what some call a fundamental component of the Web. It's not the first time the IT world has seen this kind of case. Deep linking has been a subject for debate ever since Al Gore, er, since the beginning of the Internet.

When a Web site offers links that connect directly to pages deep within other sites—that's deep linking. Those who practice it—

which is to say just about everyone who has a Web site—argue that it is an essential quality of the Internet. If the Web's creators hadn't wanted linking, "they would have called it the World Wide

Straight Line," says Avi Adelman, who operates BarkingDogs.org. Adelman has received a cease-and-desist letter from Belo Corporation, the parent company for the *Dallas Morning News*. Adelman's site contains deep links to articles in the *News*.

In 1997, Ticketmaster.com filed a suit against Microsoft for deep linking, the first major case of its kind in the United States. Microsoft's Sidewalk Web guides were deep linking to information on Ticketmaster's Web site. At the same time, CitySearch, a rival Web guide service, had signed an agreement to pay for the same links that Microsoft was using for free. The case was settled in 1999, and Microsoft agreed to only link to Ticketmaster's home page. The same year, Ticketmaster filed a similar suit against Tickets.com, saying that their deep links were violating the U.S. Copyright Act. In this case, Los Angeles District Judge Harry Hupp ruled that deep linking does not violate the Copyright Act, and compared it to using the card catalog at a library. It's over, many concluded with a sigh of relief, deep linking is okay. That is, until the decision came down from Denmark.

Maybe the problem isn't with deep linking, but with the character of the Internet itself. That's what Margaret Kubiszyn seems to be suggesting when she says that "this is the



If the creators of the Web hadn't wanted linking..

```
<ScriptParam name="ResultPage"
value="Test1AddedForm.xml" />
</TestScript>
</AcceptanceTest>
</TestSuite>
```

The "CreateScript.xml" file itself is as follows:

```
<?xml version="1.0" encoding="UTF-8" ?>
<ScriptBlock>
  <DatabaseState>
    <TestScript
name="#DatabaseScript#" />
    <DatabaseTable name="T_SEQUENCES">
      <DatabaseUpdate
where="SEQUENCE_NAME=' COMPOSERS' ">
        <UpdateField
name="SEQUENCE_VALUE" value="-1000"></
UpdateField>
      </DatabaseUpdate>
    </DatabaseTable>
  </DatabaseState>
  <MenuClick itemText="Browse Composers"
pageDescription="#ComposerListing#"
preTranslation="ComposerBrowserTransform.xml"></
MenuClick>
```

```
<PageClick itemText="Add Composer"
pageDescription="EmptyAddScreen.xml"
preTranslation="ComposerInformationTransform.xml" />
>
  <SubmitPage
pageDescription="#ResultPage#"
preTranslation="ComposerInformation
Transform.xml">
    <InputValue name="DisplayName"
value="#DisplayName#" />
    <InputValue name="ActualName"
value="#ActualName#" />
    <InputValue name="DateOfBirth"
value="#Born#" />
    <InputValue name="DateOfDeath"
value="#Died#" />
  </SubmitPage>
</ScriptBlock>
```

When the test is run, it will call yet another test script (defined by a parameter) to set up the database, then click on Browse Composers, checking the results against the HTML generated by the EmptyComposerListingPage.xml data file and the "ComposerBrowserTransform.xml" XSL. It will then click the link to add a composer and submit the resulting page with the given input values. When the test finishes, it restores the original state of

same conflict we see over and over in dealing with Internet issues—the free-wheeling, ‘anything goes’ cyber-culture fighting the evolution of the Internet into a commercial medium for companies intent on protecting their brands and corporate images.” Kubiszyn, who is a member of the Editorial Board at GigaLaw.com and practices patent, trademark, copyright, and computer law at a Birmingham, Alabama, law firm, discusses the need for definitive guidelines regarding different types of linking. But it doesn’t seem like there are going to be any for a while.

One argument used by linkers is that linking is just another part of the free nature of the Web, and that fact, by definition, makes it unique. While owners of on-line publications and other major sites want to disallow any links except those that take you to their homepages, usability expert Jakob Nielsen sings the praises of e-commerce deep linking, calling it a “hot lead to visit your Web site.” He wrote that deep linking improves usability because it takes customers directly to the information they want, rather than taking them to your homepage and making them hunt for it. He also notes that visitors are likely to use search engines like Google or Yahoo to locate deep links to specific topics anyway.

“I think maybe it’s a lesson in planning your Web site. You need

to find a technological solution and not rely on a legal one,” suggests Doug Isenberg, an Internet law attorney in Atlanta. Nielsen and other Web experts agree.

If you absolutely do not want a page deep linked, Nielsen recommends using a “noindex” meta-tag on that page. Dave Winer of DaveNet suggests using something similar to robots.txt, or the Robot Exclusion Protocol, which, in the early days of the Internet, was the accepted way to prevent scanning by Web “crawlers.” He says that the “professional and respectful way to let the outside world know that you don’t want their flow” would be a deepLinks.txt file that states that the page containing the file should not be deep linked. Other methods are password-protected or paid-customer-only pages, and scanning incoming hits and blocking those that come from unauthorized sources.

The battle isn’t over yet, but chances are the character of the Web is not in jeopardy. Nobody really wants to turn it into a straight line, do they? ↵



...they would have called it the World Wide Straight Line.

any database tables that were modified inside the DatabaseState element.

A Solution for Everyone

We tried many forms of acceptance testing, from manual tests to fully automated systems. The manual tests were unsatisfactory all around, they were subject to differing interpretations and expensive to execute. The hand-coded tests reduced the costs of running the tests but increased the costs of creating them. The developers, knowing how often it saved them from making unintentional changes to the application’s functionality, felt that they were very worthwhile, but this value was somewhat hidden from the customer. It also left too much of the interpretation of the tests in the hands of the developers.

We also tried a commercial tool for acceptance testing. This would have helped the customer code the tests himself, but the nature of the tool meant that the tests would have to be written after the stories were completed, thus depriving the developers of the opportunity to run the tests on the current story.

Avignon is an attempt to provide a satisfactory solution to both the customer and the developers. The use of XML/XSL has kept the cost of implementing Avignon quite low, yet it gives the customer a relatively easy way to specify tests in advance without ambiguity. This allows the developers to work with confidence that they’re aiming for the right target and know how far off the mark

they are while they’re coding. In the near future I’d expect to see Avignon brought into the realm of testing more traditional user interfaces and perhaps even take the burden of UI coding off the shoulders of developers. ↵

Reprinted with permission from Don Wells and Laurie Williams, editors, Extreme Programming and Agile Methods - XP/Agile Universe 2002. Second XP Universe and First Agile Universe Conference, Chicago, IL, USA. August 4-7, 2002. Proceedings. Lecture Notes in Computer Science Vol. 2418

References

- ¹ *Extreme Programming Explained*, Kent Beck (Addison-Wesley, 2000)
- ² For more information about QARun, see Compuware’s web site: <http://www.compuware.com/products/qacenter/qarun/detail.htm>.
- ³ Also expressed in: *Extreme Programming Installed*, Jeffries, Anderson and Hendrickson (Addison-Wesley, 2001)
- ⁴ For more information about HTTPUnit, see <http://www.httpunit.org>.
- ⁵ For more information about XPath, see <http://www.w3c.org/TR/xpath>.
- ⁶ For more information about XSL, see <http://www.w3c.org/Style/XSL>. Neil Bradley’s *The XSL Companion* (Addison-Wesley, 2000) is also a good introduction.

Hopper was instrumental in the development of COBOL (common business-oriented language), the first programming language to use English words instead of number code. The first step was her development of the compiler, which converts English symbols into machine code, something she said she did because she was “lazy.” Whatever the reason, it was a vital development for computers. “I felt that more people should be able to use the computer and that they should be able to talk to it in plain English,” she said of her effort to develop COBOL.

NOLA’s project manager for the DOE Oak Ridge Team, David Stillwagon, met Lt. Hopper on assignment with the Navy in 1965. “She was working at the Defense Intelligence Agency. I’m not sure if she was stationed there or on special assignment,” Mr. Stillwagon said. “She claimed she was the ‘Mother of Data Processing in the Navy.’” He remembers that she smoked a corncob pipe. “In those days smoking was permitted in the office space.” He has visited her memorial at what was then the Data Processing Center in San Diego, California.



Capt. Grace Hopper at the keyboard, August 1976.

In 1983, Lt. Hopper was promoted by presidential appointment to the rank of commodore. When the rank was merged with that of Rear Admiral, she became one of the first women to hold this rank. She retired from the Navy in 1986, and at 80 years of age was the oldest active military officer at the time. She worked as a senior consultant to Digital Equipment Corporation until shortly before her death in 1992.

Grace Murray Hopper was one of the first software engineers—and one of the only women—to hold the rank of admiral. The Data Processing Management Association honored her with the first Computer Science Man-of-the-Year Award. She was the first U.S. citizen and the first woman to be a Distinguished Fellow of the British Computer Society. She

received 72 honorary degrees from universities across the country. The Navy christened a ship in her honor: The *Amazing Grace*. She was awarded the National Medal of Technology, the nation’s highest honor in technology and engineering. She was also awarded the Defense Distinguished Service Medal, the Department of Defense’s highest honor. ↵



NOLA
COMPUTER
SERVICES
3535 Canal Street
New Orleans, LA 70119-6157

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

