IMPROVING CODE WITH STATIC ANALYSIS


Web 2.0!
AJAX Meets JavaServer Faces

27 Must-Have Eclipse Plug-Ins
Implementing XML and Service Oriented Architecture doesn’t mean starting from scratch. Why not integrate and modernize application functions as you need them in Eclipse?

**NXTware ESB** provides just-in-time modernization from within an Eclipse development environment. Lower costs and reduce risk by creating SOA interfaces for the business functions you need, when you need them, with **NXTware ESB** for Eclipse.

Learn more about Enterprise Evolution and NXTware SOA for Eclipse. Call (866) 493-4224 or visit www.ecubesystems.com.
4th Annual Gathering of the Eclipse Community

Attend EclipseCon 2007
EclipseCon is the premier technical and user conference focusing on the power of the Eclipse platform. From implementers to users, and everyone in between, if you are using, building, or considering Eclipse, EclipseCon is the conference you need to attend.

Over 200 Sessions and Tutorials Including:
- Business
- C/C++ Development
- Database Development
- Industry Vertical
- Java Development
- Mobile and Embedded
- Modeling
- OSGi
- Project Mashups
- Reporting
- Rich Client Platform
- SOA Development
- Test and Performance
- Technology and Scripting
- Tools
- Web Development

Keynotes from:

Scott Adams
Robert Lefkowitz
Dr. Herbert Thompson

This is your opportunity to get in-depth technical information from the Eclipse experts, learn the latest tips and techniques for using the tools, network with fellow enthusiasts and experience the breadth and depth of the Eclipse community. Attending EclipseCon will expand your knowledge and make Eclipse work better for you.

March 5th - 8th
Santa Clara, California

Register at: www.eclipsecon.org
Java on the desktop is back!

RCP Developer™

Instantiations RCP Developer harnesses the power of the Eclipse Rich Client Platform. Quickly create innovative desktop applications that are rich, cross-platform, and high-performance. The vast expertise of the Eclipse developer community and the experience of Instantiations lets you...

STAND ON THE SHOULDERS OF GIANTS

Create blazing fast rich Java clients.
Tap into the ultra high performance Eclipse graphical user interface libraries and ensure a native look and feel.

Build rich Java client applications faster.
Focus on creating business specific functionality rather than reinventing the low-level logic required for applications to run.

Deliver a more consistent user experience.
Tame the most demanding user interface requirements and reliably deliver a rich and consistent user experience across all applications.

Fully exploit the Eclipse Rich Client Platform.
“The Eclipse Rich Client Platform is the leading framework for creating Java client applications. RCP Developer is the first product to bring comprehensive application construction, GUI testing and packaging of rich-client applications to Eclipse RCP.”
—Mike Milinkovich, executive director of the Eclipse Foundation

Download a risk-free trial copy:
www.instantiations.com/rcpdeveloper

RCP Developer™

Design. Quickly create views, editors, perspectives, and preference pages for your applications. Intuitive visual design and access to the high-performance SWT GUI framework of Eclipse RCP.

Test. Automatically test the GUIs that comprise Eclipse RCP applications. Minimize the need to hand-code GUI test cases. Automated recording, test generation, assertion coding and playback facilities.

Deploy. Streamline build and deployment for Eclipse RCP applications. Assemble application elements into a single package. Quickly create and maintain high-quality Eclipse RCP installations.

www.instantiations.com 1-800-808-3737
The leader in Eclipse RCP development tools
FEATURES

18 COVER STORY
27 Must-Have Eclipse Plug-Ins
Rick Wayne takes you on a scenic tour of his favorite add-ins, covering language support, modeling, UI design, integration, frameworks and much more.

You can leverage Web services using the Eclipse Web Tools Platform, as Christopher Judd demonstrates.

32 Improving Code With Static Analysis
Steve Gütz teaches you how to defeat code defects by leveraging Eclipse’s Test and Performance Tools Platform.

38 AJAX Meets JavaServer Faces
Max Katz shows how you can use component-based development for building “Web 2.0”-style rich Internet applications.

DEPARTMENTS

7 Context
Happy birthday, Eclipse! It’s been five years, and look how much the platform and community have grown.

8 Launchpad
The latest tools, technologies and plug-ins for Eclipse developers.

10 Foundation
Good news on adoption of the Rich Client Platform, and a first glance toward Europa.

11 Shoptalk
With Wolfram Workbench, Mathematica users gain a sophisticated Eclipse-based IDE.

13 Projects
The Data Tools Platform makes it easy to work with relational databases.

41 Perspective
Has the open-source Eclipse platform ruined the commercial tools market—or helped it?
Sybase WorkSpace: Do Something More Interesting

A SINGLE ENVIRONMENT OFFERS WHAT YOU NEED
Sybase WorkSpace is a Java™ toolkit offering the five most important design and development tools in an integrated easy-to-use, open source framework:
• database development
• mobile development
• web application development
• enterprise modeling
• services-oriented development

MAKES DEVELOPMENT AND DESIGN EASY TO LEARN AND USE
Using industry-leading integrated model-driven design, visual development and task-based wizards, Sybase WorkSpace automates mundane tasks and cuts the typical development tool learning curve, freeing developers to concentrate on what’s important—business logic.

FREEDOM TO CUSTOMIZE BASED ON YOUR BUSINESS NEEDS
WorkSpace’s modular packaging allows enterprises the flexibility to decide how to assign business critical tasks to developers, rather than requiring them to fit into ill-fitting “roles” predetermined by a vendor. You buy only the pieces you need, enabling you to customize your environment as you see fit.

With Sybase WorkSpace, you’re finally free to do the design and development that’s interesting to you. For more information and to download White Papers and an evaluation copy, visit www.sybase.com/workspace
Happy Birthday, Eclipse!

It’s been five years since IBM launched Eclipse as an open source project. The technology had been incubating inside Big Blue since the late 1990s as a next-generation Java IDE, but IBM’s announcement that it was open-sourcing Eclipse, on Nov. 7, 2001, set the platform on the path to super-accelerated growth. Today, Eclipse is second only to Microsoft’s Visual Studio in adoption, surpassing Sun’s NetBeans and Borland’s JBuilder in the Java space.

IBM’s commitment to Eclipse was further demonstrated by its willingness to divest itself of its intellectual property and form the independent Eclipse Foundation in February 2004. The unprecedented moves toward open source and open governance launched Eclipse into orbit.

WHERE NO STANDARD IDE HAS GONE BEFORE…

But that’s not what’s sustained it there. The Eclipse Foundation and its active members have continued to broaden the platform’s charter. They’ve introduced Eclipse into new areas, going where no standard IDE has gone before. To name just a few: The Rich Client Platform has taken on a life of its own. The C/C++ Development Tool project showed that a Java IDE doesn’t have to be limited to just Java. The Device Software Development Platform helped programming think inside the box, as did the Embedded Rich Client Platform.

As Eclipse reaches its fifth year, the organization has a number of fascinating new projects that are either just setting sail or making progress. Some are making more progress than others, but that’s what happens in a community.

One new project is COSMOS, which extends the Eclipse platform outside of software development and into more general IT. “The mission of the Eclipse COSMOS Project is to build generic, extensible, standards-based components for a tools platform upon which software developers can create specialized, differentiated and interoperable offerings of tools for system management.” That’s a new direction for the Foundation, driven by companies like IBM, Cisco, Intel and Compuware.

Another advance is the Rich Server Platform User Interface Framework, or RSP-UI. (Gotta love those Eclipse acronyms!) This project is conceptually similar to the Rich Client Platform, except that it allows the creation of pluggable, componentized, server-side applications, where plug-ins communicate with each other through extension points and OSGi services. This project hasn’t made much progress lately, but the concept is fascinating.

A more active project is the Subversive Project, which seeks to create a Subversion plug-in for Eclipse with functionality similar to the Eclipse CVS Team project. If the project is successful, the team hopes that Subversion support will become a core Eclipse feature. Polarian Software, which sells application life-cycle tools for Subversion, is running the project. So far, it’s on track for release in 2007.

The first five years of Eclipse have been exciting—look how far the technology, and the organization, have come in a short time. I can’t wait to see what happens next.
**GWT Designer Delivers AJAX GUI for Google Web Tool**

Instantiations is offering GWT Designer, a new tool for AJAX Web applications using the Google Web Tool. GWT Designer is built on WindowBuilder Pro, the company’s software for creating Java graphical user-interface applications for Linux, Mac OS X and Windows. GWT Designer provides developers with a rich WYSIWYG GUI construction environment. Developers simply drag-and-drop composites, layouts and controls when building user interfaces, which can be tested and deployed using the Java-to-JavaScript compiler provided by Google. GWT Designer also includes bidirectional code generation, a WYSIWYG designer and Cascading Style Sheet support. Its GWT application launch system has a shortcut for fast launching using the popup menu or editor hot key. It also has intelligent refactoring, where renaming or moving a module class or remote service updates the module XML file and renames any associated Async and Impl classes. The software costs US$39 for an annual subscription. www.instantiations.com

**Real-Time IDE for Enea’s OSE Leverages Eclipse, CDT**

Optima is a new Eclipse-based IDE for OSE, a time-time operating system from Enea. Optima, which costs $3,000 per seat, uses Eclipse 3.1.2 and C/C++ Development 3.1.1 to provide system-level browsing, debugging, profiling and analysis tools for large-scale distributed systems spanning multiple processors and operating systems. Optima uses the Eclipse Workbench and Workspace concept for building images and managing projects, and provides plug-ins for most software configuration management tools. The CDT includes a C/C++ editor with language-aware highlighting and code completion, a C/C++ debugger, a C/C++ launcher, a parser and indexer, a search engine, content assist and a Makefile editor and builder. The company builds on Eclipse with several OSE-aware plug-ins that help developers solve complex system-level problems in their application code. These plug-ins include a system browser for viewing and manipulating OSE system objects, a pool profiler for browsing and analyzing system memory utilization, a run-mode debugger with a program launcher (loader), and a help viewer for viewing OSE documents and references. All Optima plug-ins support fully distributed debugging, which enables any target CPU in a connected network to be accessed without the need for a direct connection. www.enea.com

**GUIdancer Tests Without Coding**

GUIdancer is an Eclipse-based environment from Bredex for testing Java/Swing application GUIs that the company claims requires no coding. New in version 1.1, released last week, is the ability to execute batch tests from a command line, support for testing multilingual applications using the same test components, new capabilities for trees and tables, and an improved user interface. First released in January as a preview, GUIdancer 1.1 for Linux, Mac OS X, Solaris and Windows also now includes an API to extend the tool—without code—for applications that use non-Java/Swing components and actions. To avoid the need for programming, GUIdancer employs a so-called specification concept, which creates modular, reusable tests from source code before an application is completed. On the desktop, the program requires Java 1.5, which is included with the Windows download. The server component runs on Java 1.3. Demo licenses are free; named-user licenses cost €980, plus €245 of annual maintenance. www.guIdancer.com

**SHORT TAKES**

**Stylebase** is a new open-source tool for maintaining a reuse repository for architectural models and design patterns. The software and the repository assist developers in applying a quality-driven architecture design to software engineering in a way that promotes software reuse. The tool can be used in three ways: as an electronic library, where an architect can browse the stylebase as a pattern catalogue; for model evaluation; where the architect detects which patterns have been used in an architecture model and then checks from the stylebase which quality-attributes are associated with these patterns; and for constructing a new architecture model, where the architect searches the knowledge base according to the desired quality characteristics. stylebase.sourceforge.net

**GNATbench** for Eclipse is a plug-in that provides editing, browsing and building features for Ada (including Ada 2005) development using AdaCore’s GNAT Pro toolset on the Eclipse platform. The builder produces executables for native systems and embedded processors (in the context of Wind River’s Workbench), and the debugger supports both native and embedded system debugging. GNATbench includes an Ada-specific editor, code assistance for Ada, code browsing, a project explorer, an integrated builder and an integrated debugger. The company offers a separate Ada plug-in for Wind River’s VxWorks, but this plug-in produces executables for native systems, rather than embedded processors; similarly, the debugger supports native system debugging. www.adacore.com
Remember when the sky was the limit?

With Intel® Software Development Products, the Swinburne Center for Astrophysics is showing today’s kids a universe filled with unlimited possibilities.

THREADED APPLICATIONS HELP UNRAVEL THE ORIGINS OF THE UNIVERSE.
Take advantage of the power behind multi-core processors by introducing threading to your applications. Threading allows you to use hardware parallelism to improve application speeds. Intel® Software Development Products are on the leading edge of threading technology, giving you the opportunity to discover the performance potential you need. So whether you create applications that model the solar system or enable a gaming system, Intel Software Development Products give your applications the power of parallelism.

Good News on RCP Adoption

A recent Evans Data survey indicated strong growth in the awareness and adoption of the Eclipse Rich Client Platform (RCP). A survey of the Eclipse user community indicated that almost 23 percent of the respondents were using Eclipse RCP. This is a 130 percent increase over a previous 2005 study. More importantly, the survey indicates strong growth and momentum for future RCP usage, with 68 percent of the respondents indicating they will be using it within the next six months. A great sign of a strong future for Eclipse RCP.

ECLIPSE EVENTS
Thanks to BZ Media, the Eclipse community has a great new Eclipse conference to attend. The second annual Eclipse World conference was held Sept. 6–8, 2006, in Boston. By all accounts it was a great success and a great place to get solid technical content on different Eclipse projects. We look forward to the EclipseWorld 2007, which will be held Sept. 6–8 in Reston, Virginia.

The Eclipse Foundation also launched a new European event in Germany, called Eclipse Summit Europe. It was held Oct. 11–12 in Esslingen, Germany, and attracted 300 Eclipse enthusiasts. The Summit featured four technical symposia covering topics such as modeling, rich client applications, server-side Eclipse and test-driven development for the embedded systems space.

MAINTENANCE RELEASE FOR THE ECLIPSE CALLISTO PROJECTS
A maintenance release of the Callisto projects was made available on September 29, 2006. All 10 projects that participated in the June Callisto release coordinated the availability of their bug-fix maintenance release for this date. Users can go to the Eclipse update site to download Eclipse 3.2.1 and the new releases of the other projects.

IN 2007, IT’S EUROPA!
The planning has started for the next year’s coordinated release train, called Europa. Like Callisto, Europa will feature a coordinated release of the major Eclipse projects. The plans still need to be completed, but we expect the date to be late June 2007. The list of projects is expected to include the entire Callisto project, plus additional ones. Stay tuned for more details.

NEW MEMBERS OF THE ECLIPSE FOUNDATION
We would like welcome Simula Labs as a new strategic developer of the Eclipse Foundation. In addition, Sony Ericsson Mobile, ARM Limited, Active Grid and KPIT Cummins have joined as add-in providers. The Eclipse Foundation now has 142 member organizations and more than 750 committers.

PROJECT UPDATES
Here is a summary of project reviews that have arisen so far in 2006. We hope to make this a regular feature of this column. It is based on the RSS feed at www.eclipse.org/projects/reviews-rss.php.

<table>
<thead>
<tr>
<th>PROJECT</th>
<th>REVIEW DATE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parallel Tools Platform 1.0 Released Approved</td>
<td>Aug. 15</td>
</tr>
<tr>
<td>ECM Rich Client Platform [Apogee] Project Approved</td>
<td>Aug. 15</td>
</tr>
<tr>
<td>Google Summer of Code Project Created and Provisioned</td>
<td>Aug. 18</td>
</tr>
<tr>
<td>SVN Team Provider Proposal Updated</td>
<td>Aug. 20</td>
</tr>
<tr>
<td>Aperi Storage Management Project Created and Provisioned</td>
<td>Aug. 24</td>
</tr>
<tr>
<td>g-Eclipse Proposal Posted</td>
<td>Aug. 25</td>
</tr>
<tr>
<td>Embedded Rich Client Platform 1.0 Released Approved</td>
<td>Sept. 11</td>
</tr>
<tr>
<td>Eclipse Orbit Project Created and Provisioned</td>
<td>Sept. 22</td>
</tr>
<tr>
<td>COSMOS Proposal Updated</td>
<td>Sept. 28</td>
</tr>
<tr>
<td>Eclipse Platform 3.2 Release Approved</td>
<td>Oct. 2</td>
</tr>
<tr>
<td>Subversive Project Proposal Updated</td>
<td>Oct. 5</td>
</tr>
<tr>
<td>ECM Rich Client Platform [Apogee] Project Created and Provisioned</td>
<td>Oct. 6</td>
</tr>
<tr>
<td>g-Eclipse Creation Review</td>
<td>Oct. 15</td>
</tr>
<tr>
<td>Model-to-Text Transformation Proposal Posted</td>
<td>Oct. 16</td>
</tr>
<tr>
<td>AspectJ Development Tools 1.4.1/1.5.3 Release Review</td>
<td>Oct. 17</td>
</tr>
<tr>
<td>Model Development Tools Project Created and Provisioned</td>
<td>Oct. 18</td>
</tr>
<tr>
<td>Callisto Simultaneous Release Approved</td>
<td>Oct. 19</td>
</tr>
<tr>
<td>CDT 3.1 Release Approved</td>
<td>Oct. 19</td>
</tr>
</tbody>
</table>
It’s a given that any reader of Eclipse Review sees Eclipse as a top-notch development environment. Its extensibility via plug-ins is an added benefit. Many also know that it can be used a front end or even the basis of an application. While Wolfram Research doesn’t go as far as using the Eclipse RCP, it does make use of the first three options, making the company somewhat of a new power user in the community of Eclipse development.

Wolfram Research (www.wolfram.com), based in Champaign, Ill., is the developer of Mathematica, a well-known software application that integrates a numeric and symbolic computational engine, graphics system, programming language and documentation system along with connectivity to other applications.

Mathematica can handle complex symbolic calculations that involve millions of terms; load, analyze and provide visualization of data; solve differential equations and minimization problems numerically or symbolically; perform numerical modeling and simulations ranging from simple control systems to galaxy collisions, financial derivatives, complex biological systems, chemical reactions, environmental impact studies and magnetic fields in particle accelerators; and facilitate rapid application development for engineering companies and financial institutions.

Recently, Wolfram released a new product called Wolfram Workbench, a version of the product based on Eclipse. Users who currently use Mathematica now have a means of customizing the application via the functionality inherent in Eclipse, and current Eclipse users now have the ability to use Mathematica as a set of plug-ins in their own Eclipse installations, offering them mathematical computation capabilities that can be altered to suit their needs. They can also use Workbench to build their own technical applications, whether they’re addressing engineering, science, finance or educational markets.

ECLIPSE AS AN IDE… FOR ECLIPSE!

In addition to offering the Eclipse version of Mathematica and its capabilities in a plug-in form, Wolfram took advantage of another obvious Eclipse capability: using it as an IDE. In other words, the company used Eclipse to develop for Eclipse.

“It would be possible to build the plug-ins and Eclipse version of Mathematica without using Eclipse, but it would be enormously inconvenient,” says Tom Wickham-Jones, director of kernel technology and lead Workbench developer. “One of the key features of Eclipse is that it provides a great development environment for building Eclipse tools. Eclipse as an IDE is a wonderful tool in itself.”

Wickham-Jones believes that one of Eclipse’s strengths is its ability to support languages other than Java. “Eclipse has great application development mechanisms and useful APIs and engineering. It’s very suitable for building extra plug-ins that add support for languages other than Java. It’s quite common for people to build support for Python, Perl and other languages. What we’ve done is to build support for Mathematica.”

Mathematica is built using a heterogeneous collection of programming languages. Some of it is built in C, some in Java, and lot of it in the Mathematica language itself. The application really has no specific development environment; for the Mathematica functionality itself, there is still no alternative but to use Wolfram’s own custom programming language. For developing other parts of the product, the company has used a variety of languages and IDEs such as C++ and Visual Studio. However, because Mathematica runs on many platforms, Eclipse offers the type of flexible support that’s important to many of its users.

In the future, Wolfram is looking toward making use of some of the newer Eclipse tools that it sees as very valuable. “We hope to use some of the more recent plug-ins to build more tools for Mathematica,” says Wickham-Jones. “All of these plug-ins have their own extension mecha-
nisms. The tools we’re looking at in particular are the Web Tools Platform and C/C++ Development Tooling. To build tools for the Web version of Mathematica, we would provide users with Mathematica WTP tools. We could also use CDT to make use of Mathematica’s own extension mechanism to hook into C or C++ code.”

Wickham-Jones believes that CDT will become especially important to Mathematica users. “CDT is relatively new, but it’s picking up steam. Having a heterogeneous language system under one IDE is pretty appealing. That’s relevant to Java programmers because I think they’re much more likely to be proficient in another language in addition to Java than, say, Visual Basic programmers. CDT is another universe of programming languages. It could even help Mathematica developers build support for Fortran, which is very important for programmers dealing with mathematics. It’s somewhat specialized, but many large research labs have a huge investment in the language.”

The fact that Eclipse is an open source environment can also add value to developers using Wolfram Workbench because Eclipse is also a sort of how-to manual on Eclipse. If something doesn’t work, it’s easy to run the code through a debugger, see your source code and step into the Eclipse APIs to determine why things aren’t working the way you expected—a useful way to learn things that aren’t documented.

Wickham-Jones emphasizes that the feature provides the hard-core inner knowledge that’s usually reserved for people who have been programming for many years. Typically, commercial tools don’t offer that capability, but Wolfram provides some of its own source code for exactly the same reason—instructing its users.

The Wolfram Workbench is an IDE for Wolfram products that is actually a version of Eclipse that has been customized by adding tools for working with Mathematica and other Wolfram technologies. These tools are all built, as are all Eclipse tools, with Eclipse plug-ins—the Eclipse extension mechanism.

Wickham-Jones notes that Eclipse plug-ins have access to a large and rich library of functionality that can take advantage of many useful features such as encapsulation, versioning, localization, a help system and Web-based updating. In addition to its library and components, other key Eclipse features that Wolfram took advantage of include the ease with which plug-ins can be built.

“Eclipse contains a large number of wizards and tools for building plug-ins,” Wickham-Jones notes. “As you develop, you can easily launch another installation of the environment that uses your tools and allows you to track down problems. The environment makes it easy to create and run unit tests, which can also boost the quality of your application.”

A WORLD OF FEATURES AND FUNCTIONS

Wolfram sees a world of Eclipse features that complement its products—and Mathematica functions that could be of use to Eclipse users. One avenue the company is investigating is enhanced refactoring. For example, people conducting research often use Eclipse as sort of a vehicle for implementing their ideas.

Wickham-Jones doesn’t think it’s common to use graph theory techniques by themselves in Eclipse because it’s difficult to support large source files that use a graph to represent the files, functions and components that relate to other components. He states that it will soon be possible to take graphs from source code and use the graph theory features in Mathematica to do manipulations on these graphs.

Graph manipulation works well in Mathematica, and providing this functionality to Eclipse users could potentially be very helpful. The application also offers graph layout tools that can support tens of thousands of nodes. Future Eclipse functionality may only be on the drawing board for now, but the director of kernel technology sees many exciting possibilities.

Because Wolfram allows users of its Workbench product to use it in stand-alone mode or as a set of plug-ins, Wickham-Jones says Workbench provides another important feature. “Eclipse makes it very easy to produce an application that has your own look and feel rather than the Eclipse look and feel. This is an important step in giving your application a fully professional appearance, whether you’re simply using our products or using our set of plug-ins to add functionality to your own. We took advantage of that feature and assume that users of our plug-ins will, as well.”

Any company or institution that requires extensive mathematical functionality now has a tool that they can use in Wolfram Workbench. At the same time, we once again see that Eclipse can often be a means to an end, whether it’s used as an IDE or an extension mechanism for other products.
Dealing with data that’s stored in an outdated relational form is often the bane of a developer’s existence. However, that data is typically still valuable and accessing it is a necessity. Having an intuitive and relatively easy-to-use means of making it available is the goal of the Data Tools Platform. The three founding projects comprising DTP—Model Base, Connectivity and SQL Development Tools—meet the platform goals that their names imply in the DTP 0.9 release, and form what we’ll call the DTP core.

While frameworks and tools are the necessary foundation, you’ll have trouble leveraging DTP (www.eclipse.org/datatools) directly without additional components supporting your specific data source. While you could follow the Apache Derby specializations provided as part of DTP, that’s neither practical nor desired. Instead, you’d probably rather have specialized data source support close at hand. That’s where the DTP Enablement project comes in: It seeks to offer as many open-source data sources as possible. The Enablement project will serve as a clearinghouse for DTP data source support. Here, we’ll explain the initial steps you’ll use to access relational data sources using DTP.

LEVEL OF INTEGRATION
A data source can integrate with DTP on several levels. Depending on your requirements and data source, it might make sense to support all of these levels, or only some of them. Each DTP core project exposes API and (where relevant) extension points. These two elements are the touchpoints for DTP integration. You’ll want to survey the opportunities that DTP gives for extension, noting the places that are useful for your data source.

The first thing to consider is whether the data source’s domain is represented in an existing DTP Model Base component. As of DTP 0.9, the Model Base components are all EMF-based domain models for relational data sources. If your data source is relational, it makes sense to specialize the Model Base components for your case, since this avoids duplicating effort and offers model-driven representation of the relational domain.

Each data source has a channel, or connection, through which other components communicate with it. Typically, you’ll need a driver and configuration information to gain access to that data source. Each connection is likely to have different configuration settings, while the location and default values for the driver are usually constant across connections. In DTP Connectivity, these requirements are met by the driver template and connection management frameworks, along with their associated tools.

Once you establish the connection and populate a domain model with its contents, the next step is to build tools using these capabilities. For relational data sources, DTP has the SQL Development Tools project. This project contains tools that you’d expect from an entry-level, developer-centric SQL tool, including a SQL editor, results view, the ability to execute SQL statements and so on. The SQL tools in the current DTP release work with relational data sources based on the SQL model in a generic manner.

Once you specialize a model and provide connectivity for a relational data source, you should get basic SQL tooling support by default. In many areas, you might want to adapt the SQL tools to take advantage of specific functionality. If you study the Apache Derby sample implementation in DTP 0.9, it shouldn’t be hard to understand and use them for other cases.

MODEL SPECIALIZATION
As part of the core, the Model Base project houses domain object mod-
els for use with the rest of DTP. To keep the scope manageable in the first data source releases, DTP’s designers decided to concentrate on models for relational data sources. Based on the Eclipse Modeling Framework (EMF), it includes models for the following:

- SQL: Based on the SQL 99/03 specifications, SQL is the basis for the other models
- Database Definition: For specializations not covered by the SQL Specification
- SQL Query: An abstract view of actual SQL queries
- SQL XML Query: An abstract view of XML data processed in SQL queries

Unlike many other Eclipse platform components, you’ll need to do some work to use the models with your own database. The key steps of this process are to create your database definition and expand the Ecore/Genmodel for the SQL Model.

**DATABASE DEFINITION SURVEY**

The SQL Model is a metamodel based on the SQL 99/03 specifications, which defines all database elements for the industry standard. A database definition model derives from the SQL Model, in which all detailed database model elements—including tables, columns, user-defined data types, relationships and constraints, stored procedures and their detailed parameters—are implemented.

The DTP Model Base provides a Java program to create a new database vendor document and is stored in XML Metadata Interchange (XMI) format. The Java class uses the database definition model as a base, then goes through the model element-by-element, setting specific details. It walks through items such as columns, views and triggers, specifying available data types and qualities for those items. When the Java program is run, it will create an XMI document to serve as the database vendor documentation.

**EXPOSING THE VENDOR DOCUMENT**

Once you’ve generated a database-specific XMI database vendor document, you will need to use the org.eclipse.datatools.connectivity.sqm.core.databaseDefinition extension point to expose it. You can extend the databaseDefinition extension point for org.eclipse.datatools.connectivity .sqm.core using the following definition:

```xml
<definition
  version="4.0"
  product="MySql"
  productDisplayString="%4.0_ProductString"
  versionDisplayString="%4.0_VersionString"
  file="runtime/vendors/MySql_4.0/MySql_4.0.xmi">
</definition>
```

These values specify a unique name and version string for this given vendor XML document—MySQL as the product and 4.0 as the version.

If you look at the extension point’s file property, it refers to a particular path to the XMI document in the plug-in that exposes it. The directory under runtime/vendors corresponds to [product] underscore [version,] just like the XMI file’s name, MySQL_4.0.xmi. A display string for the product and version is provided to make it more human-read-

Table 1 | Primitive types supported by the base SQL model

<table>
<thead>
<tr>
<th>Character data</th>
<th>Binary data</th>
<th>Numeric data</th>
<th>Other</th>
</tr>
</thead>
<tbody>
<tr>
<td>Character</td>
<td>Binary</td>
<td>Numeric</td>
<td>boolean</td>
</tr>
<tr>
<td>character varying</td>
<td>binary varying</td>
<td>Decimal</td>
<td>date</td>
</tr>
<tr>
<td>character large object</td>
<td>binary large object</td>
<td>Smallest</td>
<td>time</td>
</tr>
<tr>
<td>national character</td>
<td>Integer</td>
<td></td>
<td>timestamp</td>
</tr>
<tr>
<td>national character varying</td>
<td></td>
<td>Bigint</td>
<td>interval</td>
</tr>
<tr>
<td>national character large object</td>
<td></td>
<td>Float</td>
<td>datalink</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Real</td>
<td>xml</td>
</tr>
<tr>
<td></td>
<td></td>
<td>double precision</td>
<td></td>
</tr>
</tbody>
</table>
able. In this case, the 4.0.Product String and 4.0_VERSION_STRING are offered via the plugin.properties file.

**EXPANDING THE ECORE/GENMODEL**

Typically, you don’t need to alter the definition for SQL Model. However, if you do need to add new model definition to the SQL Model, complete the following steps:

- Change the SQL Model file to add your database element or create a new Ecore that references the existing SQL Model Ecore file and extend it.
- Use the Eclipse framework to generate a new EMF model.
- Generate code to create a new Database Definition Model.

For example, you may want to add an event object to the schema element in the model. You would need to create your own model object for an event and then create a unique element that extends the schema element in the SQL model.

**DEFINING A DRIVER TEMPLATE**

Access to a database is typically mediated by a driver. In the simplest case, a JDBC driver will be available for your target database. It’s worth noting here that DTP as a platform supports the driver concept in a very generic sense, so JDBC is not required. You could, in fact, use an ODBC driver, a call-level interface (CLI) or other mechanisms as a conduit to the database. The SQL model family, however, expects a JDBC interface, so you’d have to map such alternatives onto JDBC interfaces to use the models.

The first question you need to answer is whether or not you need a driver template. DTP provides a generic JDBC driver template in which the user can specify the location of the driver archive (jar) and standard properties such as connection URL, database name and so on. The disadvantage of this approach is that reasonable defaults for specific databases can’t be expressed, so users have to provide this predictable information. An advantage is that you can specify JAR file names and perhaps even locations if these can be predicted. These default suggestions make it easier for users to understand what is required for the driver template and how to locate the necessary information.

Once defined, a driver template is global to the Eclipse instance, driven by a preference page. We’re talking about templates here, which implies a set of defaults to be overridden. In fact, users create a driver definition based on a provided template. A driver definition is an instance of a driver template and is a relationship that can be understood loosely as the same as class and object in Java. The advantage of having global driver templates and definitions is that these values tend not to be specific to tools that consume them, so other tools that require access to driver definitions can use the global values. From a user’s perspective, this provides a very important benefit: It enables the definition of a specific driver once, rather than having to keep reentering the properties for each tool.

DTP provides a driver management framework, with associated preference pages and wizards, for

---

**LISTING 1: AN EXAMPLE OF CODE FOR SUPPORTING THE PRIMITIVE TYPES SHOWN IN TABLE 1**

```java
PredefinedDataTypeDefinition characterDataTypeDefinition =
    DatabaseDefinitionFactory.eINSTANCE.createPredefinedDataTypeDefinition();
characterDataTypeDefinition.setPrimitiveType(PrimitiveType.CHARACTER_LITERAL);
characterDataTypeDefinition.getName().add("CHAR");
characterDataTypeDefinition.setName("CHARACTER");
characterDataTypeDefinition.setMaxLength(254);
characterDataTypeDefinition.setKeyConstraintSupported(true);
characterDataTypeDefinition.getDefaultValueTypes().
    add(DefaultValueType.CURRENT_USER_LITERAL);
characterDataTypeDefinition.getDefaultValueTypes().
    add(DefaultValueType.NULL_LITERAL);
characterDataTypeDefinition.setLengthSupported(true);
characterDataTypeDefinition.setJdbcEnumType(1);
characterDataTypeDefinition.setJavaClassName("java.lang.String");
```

**LISTING 2: PART OF THE EXTENSION MARK-UP USED IN DTP FOR THE EXEMPLARY GENERIC JDBC DRIVER TEMPLATE**

```xml
<extension
    point="org.eclipse.datatools.connectivity.driverExtension">
  <category
    id="org.eclipse.datatools.connectivity.db.driverCategory"
    name="%org.eclipse.datatools.connectivity.db.driverCategory"/>
  <driverTemplate
    createDefault="false"
    emptyJarListIsOK="false"
    id="org.eclipse.datatools.connectivity.db.generic.genericDriverTemplate"
    jarList=""/>
  <property
    generated="false"
    id="org.eclipse.datatools.connectivity.db.driverClass"
    name="%driverClass"
    value=""
    required="true"
    visible="true"/>
</properties>
</driverTemplate>
```
managing these templates. As a DTP extender, all you need to do is contribute to the org.eclipse.datatools.connectivity.driverExtension.

Listing 2 shows part of the extension mark-up used in DTP for the exemplary generic JDBC driver template.

Without covering all details (which can be found in the API documentation provided with DTP), you’re essentially specifying a list of properties for the template, along with any known reasonable default values. Also note that there’s a “visible” attribute available for each property. This allows you to have read-only properties that are not shown to users, but consumed by downstream tools. The driver management preference pages and wizards are generated automatically based on the property values given in a template definition. Attributes such as Required are used to determine when a driver definition is considered complete.

DEFINING A CONNECTION PROFILE
A connection profile is a group of properties that acts as the particular connection type’s definition. For example, a particular database might have one or more associated driver templates (for versions, driver types and so on).

A connection profile for that database is capable of consuming that information in a selected driver definition and using it to establish a connection to the database. A connection profile is responsible for supplying a content provider to requesting tools, such as the tree-based paradigm found in the DTP Data Source Explorer (DSE).

Once again, the notion of connection profile is agnostic with respect to the actual connection type. That is, the connection profile provides an abstraction over the connection type, which gives consuming tools flexibility in interacting with heterogeneous data sources.

As with driver templates, connection profiles are managed by the
connection management framework, along with associated wizards and other UI support. Contributions to the connection management framework are made visible by the extension point org.eclipse.datatools.connectivity.connectionProfile. Listing 3 shows part of the connection profile definition for generic JDBC.

Keep in mind a few key points. The first is that a connection profile is associated with a given category defined in the category mark-up. You could reuse a category defined elsewhere, for example, to add a connection profile to the categories provided by default. The connectionProfile mark-up is simply a declarative way of setting the category and name icon in the connection profile. The key implementation comes in the connectionFactory section. A connectionFactory gives the implementation class that is responsible for creating and maintaining connections based on client requests. In this case, the JDBCConnectionFactory class provides the implementation. Being a factory, an instantiation provides, in this case, JDBCConnection objects. When the client wants to connect to the database represented by this profile, the code shown in Listing 4 is executed in JDBCConnection.

In this simple case, why do we need the overhead of a driver definition and connection profile? At first glance, these seem like overkill, but the main goal of DTP connectivity is to provide an abstraction over the notion of connection, encapsulating the implementation details so client tools can work with connections uniformly.

While the generic JDBC case is straightforward, more complex data sources will require much more processing, and the benefit to clients is the uniform abstraction delivered by the frameworks.

MAKING THE CONNECTION

Most of the work in using DTP involves creating an EMF-based model for the particulars of your database. Driver and connection support is attained by leveraging the connectivity frameworks. While the information given here is not complete in every detail, it should contain enough of a trail guide to help you navigate the Apache Derby implementation shipped with DTP 0.9 and customize it as necessary.

The DTP Enablement project will continually include more complete and complex examples for databases. These will serve a dual purpose: providing data source support close at hand to the DTP core and serving as further examples of what can be built on the Data Tools Platform.
Installing Eclipse is like moving into a new apartment—it’s spacious, intelligently laid out, boasts shiny, cleverly designed appliances... but it’s bare. To make an apartment livable, you put your stamp on it, you move in your furniture and your wall art and your game console. To make Eclipse really sing as your development environment, you need to extend the basic IDE with tools tailored to your tasks and foibles.

In short, you need to go on a plug-in hunt. You could simply grab one of the preassembled Eclipse distributions set up with whole sets of plug-ins, and there certainly are some excellent ones out there, such as MyEclipse. But there’s a lot to be said for rolling your own environment, and I’ve focused here on the a la carte approach.

It wasn’t easy to choose among the hundreds of plug-ins out there, and your mileage may vary. I’ve assembled some real stars in each of seven categories:

- **Language support**
- **UI design**
- **UML, modeling and process support**
- **Back-end integration and framework support**
- **Utilities**
- **Testing and code analysis**

A word on licensing: Free is nice, but I haven’t neglected commercially licensed tools either. Somebody is paying to write these tools, whether it’s for sale, as a corporate contribution to the community, or the legendary spare-time hacker (when I get some, I’ll become one, I always say). It costs either dollars or opportunities to develop good software like this, so you should seriously consider paying your share one way or the other. Buy, contribute money, or contribute resources like time or server space to these projects. We’re all the better for it.

Sermon over. Without further ado... the winners, if you please!

**LANGUAGE SUPPORT**

Without getting into a which-language-is-best mud-wrestling match—I know, but I’ll never get you to agree—here are some great plug-ins for widely used languages.

---

BY RICK WAYNE

Ruby Development Tools/Radrails
www.rubypeople.org and www.radrails.org
License: Common Public License / Eclipse Public License

It takes serious gumption to admit in this magazine that I code so much in Ruby. But I climbed aboard the Ruby bandwagon in 2001, when it was nothing but a red Radio Flyer squeaking by with Matz, Dave Thomas, Andy Hunt and a couple of kazoos. These days... breathes there the programmer who hasn’t been swamped by the Rails hype? If you’re intrigued by the idea of painless database-driven Web app development, grab these two plug-ins and go for it. RDT makes Eclipse aware of Ruby’s syntax, with syntax coloring and checking, an outline view, Test::Unit integration and Ruby debugging. (Refactoring is Ruby’s Achilles’ heel: In RDT, it’s limited to block comment/uncomment! And I’ll make you Java hacks a deal: You don’t bring up autocompletion, I won’t bring up checked exceptions.) Radrails provides management tools for configuring and running Rails Web applications. Neither has the laundry list of features that you’ll find in, say, J2EE tools, but in a way, that’s the point: Rails frees you from a lot of that picayune detail.

oXygen Plug-in Edition
oxgenxml.com
License: Commercial

XML is definitely here to stay, like it or loathe it (YAML, anyone?). Many of the features that make XML easy for computers to parse make it hellish for humans to edit—hence oXygen. This critter handles syntax highlighting and autocompletion, DTD and schema validation (including XML Schema and Relax NG), and features both an XSLT transformer and debugger. It also includes refactoring tools.

---

Rick Wayne (fewayne@gmail.com) is a software developer for the University of Wisconsin, where he has far too much fun building Java and Web applications. He started coding professionally in 1984, writing about it in 2000, and teaching it in 2001.
for XML and XSLT, XQuery, a built-in Subversion client and special-purpose editors for specific XML dialects like XSL:FO for PDF production or other page-formatting tasks, the Scalable Vector Graphics format (with a live preview) and the Web Services Definition Language (WSDL).

**EPIC (Eclipse Perl Integration)**

e-p-i-c.sourceforge.net

*License: CPL*

Brethren of Perl, I say unto you, suffer no longer under vi or EMACS! Step with me into the light of Eclipse (guess that’s an oxymoron). At any rate, if you’re not digging for Rubies or wrestling Pythons, the Perl world is probably your oyster, and this little plug-in offers syntax highlighting and on-the-fly checking, quick-reference documentation, expandable code templates, a debugger, an outline view and a regular-expression evaluator.

**Visual SlickEdit Plug-in**

slickedit.com

*License: Commercial*

Until the big-forehead aliens gift us with thought-driven computers, programming is going to be primarily about editing text. Deal.

Of course, one way to do so is to put the biggest, baddest text-editing tools available into your personal box, and SlickEdit is certainly one of those. I’ve been programming for dollars since 1984, and I have to tell you that this thing supports languages I’ve never even heard of. Some of the conventions are a little odd to those of you weaned on CUA (“Copy a line with a single keystroke”—huh?). But trust me on this one—it flies. Now, if they’d just ship a Macintosh version...

**PHPEclipse**

phpeclipse.de

*License: CPL*

There are plenty of dedicated PHP IDEs around, but none of them offer Eclipse’s potent combination of cross-platform compatibility and a mature toolset for so many development tasks. PHPEclipse offers syntax coloring for HTML, XML, CSS and of course PHP, with code completion for the latter. You also get a debugger and Eclipse’s highly useful outline view of your code; when it comes time for testing, you can control your test Web and database servers from within Eclipse, and see the results right in the IDE.

**UI DESIGN**

WYSIWYG Java UI design tools appeared shortly after the dinosaurs died out, but any prehistoric Java hacker watching her Visual Basic colleague at work was likely to turn a fetching shade of green. And my programming partner, the Finn, still refuses to do UI in Java. I need to get him one of these.

**WindowBuilder Pro 5.1.0**

http://windowbuilderpro.com

*License: Commercial (pared-down version is free)*

Third-gen tools like WindowBuilder Pro ease the pain of layouts, event handlers and alignment that used to be our lot. WindowBuilder Pro sports round-trip engineering of both Swing and SWT, drag-and-drop rearrangement of GridBagLayouts and internationalization support, all in pure Swing or SWT—it doesn’t add a single JAR to your application!

**Jigloo SWT/Swing GUI Builder 3.9.2**

cloudgarden.com/jigloo

*License: Free for noncommercial use*

Lock-in is beautiful if you’re a vendor, ugly if you’re a coder. Jigloo not only does round-trip GUI design, it’s specifically designed to handle UI code generated by hand or in Netbeans, JBuilder or the Visual Editor plug-in. It can convert a GUI from Swing to SWT and vice
versa, and offers a toolbar button to preview your interface.

**Visual Editor**
eclipse.org/vep/WebContent/main.php

License: EPL
The Eclipse Visual Editor Project is primarily infrastructure for generic visual editors in Eclipse, but happily ships with GUI builders for both Swing and SWT. Nothing too novel for the end user, of course; you get drag-and-drop component placement, round-tripping (the only state saved is in the Java source code, so you can edit text or whang GUI components around) and property editors for the components. New in 1.2 (the latest version at press time) are support for JFace viewers, support for Mac OS X and an XML Metadata Interchange (XMI) view for developers who want to see the internal model.

**UML, MODELING AND PROCESS SUPPORT**
Perhaps it’s heretical to proclaim in a tools article that communication trumps code-tweaking. But experience shows that when projects fail, it’s usually because the coders and stakeholders are on different wavelengths. Concision is key, and UML 2.0 is our best shot yet at a concise, accurate, widely comprehensible channel between modelers, architects and programmers. And even within the programming shop, tools can significantly lighten the day-to-day load.

**Omondo EclipseUML**
www.omondo.com

License: Commercial (single-user free version available)
Omondo’s EclipseUML supports diagrams common (class, activity) and obscure (robustness, deployment), and neatly round-trips from your code, with too many features to list here. The company’s support for XML Metadata Interchange gives us hope that modeling tools will one day be interchangeable—bravo!

**Eclipse Process Framework** and **OpenUP**
eclipse.org/epf

License: EPL
Like standards, the lovely thing about software development processes is that there are so many to choose from. The Eclipse Process Framework project, in incubator phase at press time, plans to create a minimal and extensible iterative process that’s useful out of the box, and is supported by tools integrated with Eclipse. (See Per Kroll and Kurt Sand’s article in this magazine, “A Development Library at Your Fingertips,” Summer 2006.) OpenUP/Basic, as its name implies, draws on IBM Rational’s RUP, but is aimed at smaller teams and smaller projects. This is definitely a project to watch.

**Subclipse**
subclipse.tigris.org
License: EPL
I highly respect the venerable CVS revision-control system, but let’s be honest: Subversion is just better. It’s smarter about offline work and network traffic, the revision numbers are per-commit instead of per-file, and tags and branches are cheap and simple to use. (And one use case alone motivated me to switch: Renaming folders. ‘Nuff said.) The Subclipse plug-in works very similarly to the CVS Repository Explorer and can handle repositories in local files or served up via the Subversion daemon, WebDAV, HTTP or tunneled through Secure Shell (svn+ssh). Daily operations like update, commit and compare against the latest or base rev are just a right-click away, though I admit I do keep the command-line Subversion client around for the funkier stuff.

**AccuBridge**
www.accurev.com/accubridge-eclipse.html
License: Commercial (free plug-in to SCM product)
Any SCM system stores file revisions, but AccuRev excels at the edge use cases that are problematic for CVS or Subversion: promotion, merging, change packages, even demonstration of regulatory compliance. The
_INFRASTRUCTURE LOG

_DAY 15: This project’s out of control. The development team’s trying to write apps supporting a service oriented architecture, but it’s taking forever. Gil’s resorted to giving them all coffee IVs. Now they’re on java while using JAVA. Oh, the irony.

_DAY 16: Big crisis—we’ve just run out of half-and-half!!

_DAY 18: I’ve found a better way: IBM Rational. It’s a modular software development platform based on Eclipse that helps the team model, assemble, deploy and manage service oriented architecture projects. The whole process is simpler and faster, and all our apps are flexible and reusable. The software we write today will be the software we use tomorrow. :)

The team says it’s nice to taste coffee again, but actually drinking it is sooo inefficient!

Download the IBM Software Architect Kit at: IBM.COM/TAKEBACKCONTROL/FLEXIBLE
AccuBridge plug-in lets you enjoy stream SCM without ever leaving the comfy confines of Eclipse.

**BACK-END INTEGRATION AND FRAMEWORK SUPPORT**

The new UI tools notwithstanding, Java programming today is primarily about server-based enterprise applications, and these plug-ins hook Eclipse up with the machinery that makes it run.

**Terracotta Distributed Shared Objects**

terracottatech.com/terracotta_DSO.shtml

License: Commercial (free plug-in to commercial tool)

I’m cheating a bit here. Terracotta DSO is not itself an Eclipse plug-in; it’s no little freeware utility. It’s enterprise infrastructure, a slick means of exploiting parallelism with minimal pain, scaling your J2EE applications by sharing their heaps across multiple JVMs. This Eclipse plug-in makes that process easier and more graphical, generating the necessary XML config files for you and allowing you to control the servers.

**Hibernate Synchronizer**

hibernatesynch.sourceforge.net

License: GNU Lesser General Public License

Is de-drudginator even a word? Well, it should be. The Hibernate Synchronizer certainly is one. As with any mapping framework, Hibernate’s database and code sides take some scutwork to keep in synch; the Synchronizer is designed to generate value objects, composite key objects, data access objects and such automatically whenever the mapping files change. Of course, anyone who’s used a code generator knows that the problems don’t occur the first time you use it; it’s the second, when the blamed thing overwrites all your customizations, that’s so annoying. Synchronizer cleverly gets around that by generating abstract base classes along with an extension class. Only the former is overwritten in subsequent regens, so your code in the latter stays safe.

**Hibernate Tools**

www.hibernate.org/255.html

License: LGPL

While the Synchronizer does one small task well, the Hibernate Tools do... well, basically everything else related to the Hibernate persistence framework. Not only do the Hibernate Tools bolster your ability to create configuration and mapping files, they’re round-trip—you can point the Tools at an existing database (or an existing Hibernate configuration) and get Java, config and mapping files automagically. The Hibernate Tools provide a mapping editor, a Hibernate console manager and a set of wizards for configuration and code generation.

**Spring IDE**

springide.org/project

License: Apache

When I first started learning about J2EE, I felt rather stupid. “Whoa—this is a lot of stuff.” Little did I realize that I was merely ahead of the curve (hey, that’s my story, and I’m sticking to it like white on rice).

Spring was born of frustration with J2EE complexity. It’s a full-stack application framework that’s considerably easier to use and deploy, while remaining compatible with many of the original J2EE technologies. And the Spring IDE is the Eclipse way to get on that bandwagon, offering a straightforward wizard to create Spring projects, then a set of editors and builders to extend, modify and validate the config files and beans that make up a project.

**Log4E**

log4e.jayefem.de

License: Parallel free and commercial versions

Since you’re not going to be glued to the debugger for the life of your software, you’ll occasionally need to know what it’s been up to, after the fact. So: logging. Pretty soon you’ll outgrow println statements and want to use something a little more powerful and configurable; Log4E greatly streamlines incorporation of JDK logging. Log4J or Jakarta Commons logging in your projects. You just right-click, and Log4E handles insertion of the appropriate statements; it can also automatically replace those System.out.printlns for you. It’s a tool worth using.
UTILITIES

The beauty of the Eclipse ecosystem is that it encourages these little gems of functionality and integrates them into your work.

QuickREx
bastian-bergerhoff.com/eclipse/features/web/QuickREx/toc.html
License: EPL

Perl hacks used to sneer at Java coders, parsing their way through Strings with index() and rindex(). Once we got regular expressions, of course, they went from snobbery to sympathy. Sure, regexes are powerful and succinct. Then again, so was APL, and frustrated Java programmers quickly filled up the 12-step groups.

Not only does QuickREx allow you to rapidly iterate toward a regular-expression solution on test text, it offers tab-key suggested completions, automatically escapes those pesky backslashes for you, and allows you to compile your best gems in a personal regular-expression library.

Tiger XSLT Mapper
www.axizon.com
License: Commercial

If you don’t use Extensible Stylesheet Language Transformations (XSLT) to modify XML streams, or think that it has just jumped the shark, go on to the next plug-in. (If, worse, you think XSLT so transparent that tools are redundant, the door is that-a-way, fella.) But if you need this plug-in, you almost certainly need it four hundred smackers’ worth. It’s nothing less than a point-and-click mapper for transforms.

You drag between the source XML on the left and the target XML on the right, it comes up with the transforms for you, in your choice of “optimized” or “human-readable” flavors. The reduction in your Tums budget alone might cover it.

EHEP (Eclipse Hex Editor Plug-in)
ehep.sourceforge.net
License: GNU Public License

Like Garrison Keillor’s “A Prairie Home Companion,” all of our software is sponsored by “Raw… Bits.” And sometimes, ya just gotta grit your teeth and plunge into ’em.

Programmer’s text editors like BRIEF, Epsilon and SlickEdit have had this basically forever; frankly, I was nonplussed that Eclipse lacks this feature out of the box. EHEP offers a hex view of binary files, lets you view them under a variety of encoding schemes (for example, ASCII, ISO8859_5 or MacUkraine), and supports searching for either hex or text strings.

TESTING AND CODE ANALYSIS
Test and prevent now, or fix bugs later. It’s really up to you (and your manager or customer), but “now” is a lot cheaper and faster.

Lattix LDM
lattix.com/products/products.html
License: Commercial

“Cycles of dependency” concern more than just welfare researchers. Useful code necessarily is complex—but not inevitably unmanageable!

To “help you discover, analyze, define and control your architecture,” Lattix built LDM. Peter Varhol’s recent article in this magazine, “Leaving the Crystal Ball Behind” (Summer 2006), discussed this, showing that it visualizes dependencies in a collapsible matrix, helping you stick to a layered architecture and minimize dependency cycles.
Parasoft Jtest
parasoft.com/jsp/products.jsp
License: Commercial

Combining merciless code scrutiny with automated test-case generation, I predict that Jtest will start by humbling you and finish by empowering you. It applies literally hundreds of rules (the list is customizable, of course) to a static analysis of your source code, then automatically generates test cases to wring it out, and even helps track down the hard-to-spot runtime issues.

Cenqua Clover
cenqua.com/clover
License: Commercial

If you’re using Eclipse and you’re not doing unit testing… we need to talk, because I’m interested in how you stay in business. Even if you are, and your developers proudly report creating such-and-so-many unit tests each week… how on earth do you know if they’re writing the right ones? Clover is a code coverage tool aimed at finding out the answer, and it tells you by statements, branches and methods. Not only does it give you a coverage snapshot, but it reports the trend over time; reports come out in the plug-in’s UI, in HTML, PDF or XML for ingestion into some other tool. Clover won a Productivity Award in last year’s Jolt Awards and was well regarded by the judges.

Eclipse Profiler
eclipsecolorer.sourceforge.net/index_profiler.html
License: CPL

A basic profiler plug-in, the Eclipse Profiler can display running threads and a call tree for a thread. Tooltips provide timing info; you can jump from a call-tree node to its definition in the code. You can also see profile information by class or package, filtered to zero in on the items of interest. Eclipse Profiler can do remote profiling; for example, the Tomcat heap, but you’ll need to manually set up environment variables or command-line switches on the program being run to do so.

EJ Technologies JProfiler
ej-technologies.com
License: Commercial

If you want to get fancier with your profiling and don’t mind paying for it, consider JProfiler. As with Eclipse Profiler, remote connections require a bit of command-line tweaking, but JProfiler includes wizards to hook up to common application servers, and can do tricks like analyzing a profiling session post hoc or comparing two sessions. You can slice the loaf several ways: by CPU, by thread, or by walking the heap to find allocation hotspots.

Findbugs Eclipse Plug-in
findbugs.sourceforge.net
License: LGPL

Findbugs is an open-source code analyzer—think of it as lint for Java on steroids. But there’s a crucial difference: Findbugs doesn’t deluge you with an undifferentiated wail about problems pressing and picayune. Instead, it categorizes them usefully: correctness, bad practice and style, and applies confidence ratings to its analysis, to emphasize what it’s sure is wrong. The Eclipse plug-in, of course, integrates Findbugs intelligence into the IDE, decorating your source windows with problem markers (and adding lines to the Problems view); you can control which of its rules are run against your code, and the level of hysteria you’re willing to tolerate. The rules run the gamut from the obviously broken (test for equality with floating-point numbers) through the working-but-troublesome (circular dependencies among classes) to JUnit idioms and much more.

YOUR NEW HOME

We’ve got the furniture in and the boxes unpacked—pizza-party time! (Actually, after cramming all these tools into my IDE, my living space feels distinctly overstuffed.) Your list of the best plug-ins may certainly differ from mine—in which case, we hope to hear from you—a good discussion will benefit all of us. So why not drop us a line at feedback@bzmedia.com to get the discussion rolling? Enough with the tools—back to the code!
Web services are rapidly becoming the most popular form of integration. This is primarily due to their interoperability with any language and platform that communicates over HTTP and can manipulate XML. In addition, because it communicates over HTTP, which is commonly firewall-friendly, it’s operationally easier to use than previous remote invocation technologies like CORBA, which required additional ports and protocols.

Though it previously lacked native support for producing and consuming Web services, with last year’s introduction of the Web Tools Platform, Eclipse now includes wizards and a Web Services Definition Language editor for generating Web service servers and clients, as well as server definitions for deploying Web services and tools for testing Web services, including Web Service Explorer and TCP/IP.

Here, I’ll discuss a strategy for generating and packaging Web service clients using WTP, and how to include and use the Web service client in a simple Web application.

**SAMPLE APPLICATION**

The sample application shown in Figure 1 (see page 28) is a simple dynamic Web page that a travel agent or customer might use to find information about a destination based on zip code. The application uses two .NET Web services found on the public Web services registry X Methods, located at www.xmethods.net.

The first Web service is the Weather Forecast service provided by WebserviceX.NET. A service description is available at www.webservicex.net/WeatherForecast.asmx. Later on, we’ll use the GetWeatherByZipCode service to return a seven-day weather forecast as well as place name, state code, latitude and longitude.

The second Web service is a Zip Code Information service provided by Ripe Development. A service description is available at www.ripedev.com/testService.aspx. Here, you’ll use the zipCodeToAreaCode and zipCodeToTimeZone methods to return the area code

**BY CHRISTOPHER JUDD**

Christopher Judd is president and primary consultant of Judd Solutions LLC. (www.juddsolutions.com). Co-author of “Enterprise Java Development on a Budget” (Apress, 2003) and “Pro Eclipse JST” (Apress, 2005), he also serves as an industry speaker, open source evangelist and consultant in Java, J2EE, J2ME, Web services and related technologies.
WEB SERVICE CLIENT STRATEGY

Underneath the covers, WTP 1.5 uses version 1.3 of the popular Apache Axis Web service framework found at http://ws.apache.org/axis. To develop a Web service client, WTP and Axis generate the client-stub Java files based on the WSDL, which is an XML document that describes the Web service interface. The WSDL includes exposed service-method signatures, complex data-type definitions and the URL in which to invoke the service.

Figure 2 is a graphical representation of the WeatherForecast WSDL file using the WTP’s WSDL designer. The WeatherForecastSoap service exposes the service methods of GetWeatherByZipCode and GetWeatherByPlaceName. Drilling into the output of GetWeatherByZipCode reveals the graphical presentation of the complex data type returned by GetWeatherByZipCode.

As shown in Figure 3, GetWeatherByZipCode returns a complex type of WeatherForecasts that contains attributes of PlaceName, StateCode, Latitude, Longitude and Details. This is an array of WeatherData types that contain the day and the URL to a weather map, as well as a temperature for each of the seven days in the forecast.

A common antipattern is to generate the Web service client-stub code directly into an application. This is an antipattern because the generated classes often don’t meet company code-quality and package-naming standards, and require that at least a handful of files be version controlled and managed, although they’ll never be modified. In addition, it reduces reuse, because each application invoking the service must generate and maintain its own copies of the same files.

An alternative best practice would be to treat the Web service as you would a third-party library. In the future, this wizard could be extended to support other client types—even a .NET client type.

But in the future, this wizard could be extended to support other client types—even a .NET client type.

Below the Client type is an innovative slider bar with a graphical representation for selecting how much WTP and Axis will generate. For a Web service client, the only level that applies is the first level, Develop client. Selecting any other level will produce the same results: the Web service client proxy code. The only configuration to the right and below that applies to generating the client is Web service runtime; however, just like the limited Client types at this time, the Web service runtime is currently limited to Axis.

The second page in the wizard prompts for an output folder using a drop-down box containing a list of Java source directories. Here, the Java clients will be generated. The wizard also enables the customization of namespace to package mapping if desired. If this option is selected, another page will be added to the wizard to enable mapping. By default, the wizard will use the targetNamespace attribute in the wsdl:definitions element to create a Java package for the generated Java source, but be forewarned—this often produces ugly package names.

For example, the Zip Code Information service target namespace, http://ripedev.com/xsd/ZipCodeResults.xsd,
generates the unusual package name of com.ripedev.xsd.ZipCodeResults_xsd. Note that this package name breaks common Java package-naming conventions of using lowercase names and no underscores. So, using the namespace to the package-mapping page, you can generate the client stubs and complex data types in more conventional package names. The page contains a table with two columns: The first contains the namespace in a HTTP URL format from the WSDL, and the second holds the Java package name it should map to. Use the add button below the table to add the desired mappings.

When the Weather Forecast Web service wizard wraps up, it generates specific files, as described in Table 1 (see page 30). In general, though, the wizard generates a service interface as well as a dynamic proxy and stub implementation. It also generates a service locator interface and implementation, and finally, JavaBeans/data transfer objects (DTO) for each of the complex types required by the Web service. These classes will be further discussed in the “Using the Web Service Client” section.

The Web Service Client wizard also adds the following Axis JARs to your project: axis.jar, commons-discovery-0.2.jar, jaxrpc.jar, saaj.jar, wsdl4j-1.5.1.jar and commons-logging-1.0.4.jar. This is important because these JARs must be packaged with your application to use the generated Web service client.

Before packaging your Web service client JAR, make sure you create the aforementioned readme.txt file in your project. In addition to the version of Axis, you may also want to include any namespace to package mappings you might have performed.

To create the Web service client JAR file, use the Eclipse export JAR functionality by right-clicking on your project and selecting Export > Java > JAR file. In the JAR Export wizard, uncheck the .classpath and .project files to prevent them from being included in the JAR. In addition, select a name and location for the resulting JAR file. A good naming convention is a descriptive name followed by ws-client to indicate it’s a Web service client. End the filename with a version number to accommodate future changes made to the Web service. An example filename for the Weather Forecast service is weatherforecast-ws-client-1.0.jar.

You may want to generate a source and JavaDoc JAR for your Web service client to make debugging and using the client easier.

Once the Web service client JAR has been created, you may delete the Eclipse Java project and generated code because you can always regenerate it if necessary from the WSDL located in the JAR.

**CONFIGURING WTP SERVERS**

Before you can begin creating a Web application in your workspace to use the generated and packaged Web service client, configure a WTP server runtime for your application server. Examples of server runtimes include Web container-only servers such as Apache Tomcat found at http://tomcat.apache.org or a full-blown JEE application server such as Apache Geronimo, found at http://geronimo.apache.org.

To create a server runtime configuration in your workspace, go to Window > Preferences > Server > Installed Runtime. Use the Add button to start the New Server Runtime wizard. On the first page of the wizard, select your desired application server from the list of available server runtime. Or, if you don’t see your preferred application server, use the Don’t see your application server listed? link.

The second page of the New Server Runtime wizard is specific to the application server chosen on the previous page. The Apache Tomcat 5.5 configuration is typical of most server configurations, requiring a directory where the application server has been installed, as well as a JRE to run the server.
WTP shouldn’t have any effect on your application’s runtime, so you won’t be required to deploy any WTP JARs with your application. That’s why Apache Axis is used as the Web service framework. Likewise, you must install your application server separately and configure the server runtime configuration to point to the directory using the installation directory field.

Note: Most application servers require that a JDK including a tools.jar, not a JRE, be configured in the JRE field since a JDK is required to compile JSPs. Even if your application has no JSPs, most application server admin applications do, so it’s a good practice to use an appropriate JDK for your app server. If you haven’t configured one, use the Installed JREs button to do so.

**CREATING A WEB APPLICATION**

To create a Web application, use WTP’s Dynamic Web Project wizard by selecting New > Project > Web > Dynamic Web Project. On the first page of the Dynamic Web Project wizard, provide a name and make sure the appropriate server configuration is selected in the Target Runtime. The remaining page defaults are fine for a simple Web application, so you can finish the wizard.

Upon completing the Dynamic Web Project, you are prompted to switch to the WTP J2EE perspective if you aren’t already in it. You’ll also have a new project containing a Java source directory of src, a build/classes directory for generated classes as well as a WebContent directory to contain Web artifacts defined by the Servlet specifications. The WebContent/WEB-INF is a special directory that contains files that aren’t externally addressable via HTTP when the application is deployed. By default, it contains the required Web.xml deployment descriptor and a lib directory for jars your application depends on.

**USING THE WEB SERVICE CLIENT**

Before writing the code to execute the Web services, the Web service client JARs and Axis JARs must be copied to the Web application’s WEB-INF/lib directory so they’ll be included in the Web application’s class loader at runtime. In WTP 1.5, the Axis JARs axis.jar, commons-discovery-0.2.jar, jaxrpc.jar and saaj.jar can be found in the eclipse/plugins/org.apache.axis_1.3.0.v200606181221/lib directory, and commons-logging-1.0.4.jar can be found in the eclipse/plugins/org.apache.commons_logging_1.0.4.v200606131651/lib directory. These URLs are likely to change as WTP releases new versions, so a better method for determining the location of the Axis JARs is to look at the Web service client project Java Build Path properties.

Calling a Web service from an Axis-generated class client proxy is a boilerplate three-step process, as shown in Listing 1. In the first step, you create an instance of the generated locator. In Listing 1, this is an instance of WeatherForecastLocator.

Next, you look up the service instance using the locator’s get method. The get method is overloaded to accept a java.net.URL in case you need to call it at a different URL than the default specified in the WSDL. This often happens if you call different instances of the Web service in test environments than and in production. If you do this, make sure you externalize the URL and call the getter that accepts a java.net.URL. The final step is to call the actual service method and get the results back.

The code shown in Listing 1 can be called by just about any Java code that includes the Web service client JARs and Axis JARs in its classpath. This includes Servlets, Struts Actions, JSF, EJB, Swing, unit tests and even command-line Java applications. Listing 2 demonstrates calling the Weather Forecast and Zip Code Information service from a JSP scriptlet of an index.jsp page created using the WTP JSP wizard.

**Listing 1 | Calling the Weather Forecast Web service**

// create locator
WeatherForecastLocator forcastLocator = new WeatherForecastLocator();
// use locator to look up service class
WeatherForecastSoap weatherForecastSoap = forcastLocator.getWeatherForecastSoap();
// invoke service methods
WeatherForecasts forecasts = weatherForecastSoap.getWeatherByZipCode(zipCode);
WeatherData[] details = forecasts.getDetails();
DEPLOYING A WEB APPLICATION

The easiest way to deploy and test a Web application is to right-click on the Web project and choose Run As: Run on Server. A Run on Server wizard will be invoked, prompting you to select a server to run on. The second page of the wizard will list all the Web, EJB and EAR projects in your current workspace if your server is a full JEE server, or only Web projects if your server is a Web container. Moving the projects from the Available projects list to the Configured projects list instructs WTP to deploy those applications on server startup. When you click Finished, the server is started and the application is deployed.

Once the server is started, an internal Web browser will open and navigate to your Web application. A server configuration will also be added to the Server view, which can be used to start and stop the server. The Server view can also be used to restart the server in debug mode, causing the debugger to be invoked when it hits breakpoints in any Java file or JSP.

And that’s it! Note that WTP doesn’t provide any more functionality for generating Web service clients than is available using the Apache Axis framework’s command-line utilities or Apache ANT tasks. However, its Apache Axis integration and wizards make generating and packaging a Java Web service client quicker and easier.

WTP also provides all the features necessary to build, deploy and test JEE applications that use the Web service clients it generates to consume and aggregate Web services.

---

**Listing 2** Complete index.jsp

```html
<%@ page language="java" contentType="text/html; charset=ISO-8859-1" pageEncoding="ISO-8859-1"%>

```html
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="x" uri="http://java.sun.com/jsp/jstl/xml" %>

```java
<%@page import="net.webservicex.weather.*"%>
<%@page import="com.ripedev.zipcode.*" %>

```java
<% String zipCode = request.getParameter("zipCode");
if (zipCode == null || ".equals(zipCode)) {
    zipCode = "90210";
}

// Weather
WeatherForecastLocator forcastLocator = new WeatherForecastLocator();
WeatherForecastSoap weatherForecastSoap = forcastLocator.getWeatherForecastSoap();
WeatherForecasts forecasts = weatherForecastSoap.getWeatherByZipCode(zipCode);
WeatherData[] details = forecasts.getDetails();
pageContext.setAttribute("forecasts", forecasts);
pageContext.setAttribute("details", details);
pageContext.setAttribute("zipCode", zipCode);

// Zip Code
ZipCodeLocator locator = new ZipCodeLocator();
ZipCodeSoap zipCodeSoap = locator.getZipCodeSoap();
Object areaCode = (Object)zipCodeSoap.zipCodeToAreaCode(zipCode)[0];
String timeZone = (String)zipCodeSoap.zipCodeToTimeZone(zipCode)[0];
pageContext.setAttribute("areaCode", areaCode);
pageContext.setAttribute("timeZone", timeZone);
%

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Travel Agent Zip Code Search</title>
</head>
<body>

<form action="index.jsp">
Zip code: <input type="text" name="zipCode" value="${zipCode}">
<input type="submit">
</form>

Place Names: ${forecasts.placeName}<br/>
State Code: ${forecasts.stateCode}<br/>
Area Code: ${areaCode}<br/>
TimeZone: ${timeZone}<br/>
Latitude: ${forecasts.latitude}<br/>
Longitude: ${forecasts.longitude}<br/>
Allocation Factor: ${forecasts.allocationFactor}<br/>
FIPS Code: ${forecasts.fipsCode}<br/>

<p>
7 Day Weather Forecast:<p>

<table>
<c:forEach var="detail" items="${details}"
<tr>
<td>${detail.day}</td>
<td><img src="${detail.weatherImage}" /></td>
<td>${detail.maxTemperatureF}&deg;F/${detail.minTemperatureF}&deg;F<br/>
${detail.maxTemperatureC}&deg;C/${detail.minTemperatureC}&deg;C<br/></td>
</tr>
</c:forEach>
</table>

</body>
</html>
```
Improving Code With Static Analysis

As software becomes more complex, the probability of exposing end users to program defects increases exponentially. While it was once possible to modify code and stage a code review with fellow developers, this scenario frequently misses key coding problems due to inattention to the complexities of the code.

Today, with modern software projects typically comprised of hundreds of classes and many thousands of lines of code, peer review no longer works. To help resolve some of these weaknesses in the software development process, a range of static analysis tools has evolved that automatically detect and often correct common problems in the source code.

The Eclipse Test and Performance Tools Platform has static analysis capabilities in addition to its well-known profiling and runtime testing features. You can use TPTP’s static analysis framework for performing automatic code review for Java and C++ code. It’s also a complete API for developing additional custom rules, data reporting and even entirely designing your own new forms of analysis. Let’s see how it works. But first, let’s talk about the biggest question: Why?

**WHY USE STATIC ANALYSIS?**

The real question to ask is “Why not?” Using static analysis tools is the easiest path to writing better code. Among many other studies over the past two decades, a 2005 IBM coding practice survey, both internally and of IBM Global Services customers, suggests that static analysis tools can help find 5 to 15 percent of all application defects—and some empirical studies even claim results as high as 30 percent. Regardless of the quantification of the results, one measure is certain: The earlier a problem is detected in the development cycle, the cheaper it is to correct.

If you’ve ever enrolled in a software design course, you’re already familiar with the graph in Figure 1, which shows the exponential cost of bug fixing. If a bug is detected early, the cost is minuscule; however, finding and fixing a bug after customer deployment can cost many thousands of dollars.

The easiest way to achieve early detection and correction of defects is to apply automatic tools, and what better tool than one that’s free?

**INTRODUCING THE ECLIPSE TPTP STATIC TOOLS**

In the vast static analysis landscape, you can choose from dozens of great commercial tools already available. A quick Internet search will produce a huge collection of free tools that detect many coding problems. So why do you need another?

The basic problem with all static analysis tools (and software tools in general) is that they all look different—and act differently. For example, the workflow for Parasoft’s Jtest product is quite different from Instantiations’ CodePro AnalytiX, even though both ultimately accomplish the same task. The configurations of rules and viewing of results are quite diverse between these and other products. So while it’s common to use more than one tool, there is no consistent static analysis process.

**BY STEVE GÜTZ**

Steve Gütz is a senior software developer and team lead in IBM Rational’s modeling tools group. He has been a committer on the Test & Performance Tools Platform for the past two years. Previously, he held senior management and executive positions in several public and private companies, including two of his own successful start-ups.
Additionally, among tools, the rule APIs, data exporting and reporting (if these feature exist at all) are diverse.

When developing the static analysis tool for TPTP, the development team had two main goals. First, they wanted to make all forms of static analysis consistent for the user by offering a common user interface and configuration process. They also wanted to provide a consistent and common API for static analysis tool developers to extend the framework.

Both of these themes are prevalent throughout this article, so let’s get started by installing TPTP’s static analysis capabilities into the Eclipse Workbench.

As noted, TPTP’s best feature is, of course, its price—free! You can install it seamlessly into the Eclipse IDE or almost any tool derived from the Eclipse platform. Users will benefit from the rich collection of code review rules for Java and C++, while developers can adopt the API to easily extend the rule set to enhance TPTP’s code review capabilities or to enforce internal corporate coding conventions.

GETTING STARTED WITH TPTP AND STATIC ANALYSIS

Whether you’re a developer or end user, to get started with TPTP static analysis you must meet a few requirements. First, you need an Eclipse IDE, and the easiest way to obtain this is by installing the platform’s full Callisto release. Note that you can also build up an Eclipse environment from component pieces, but this is more challenging. Regardless of which installation method you choose, you’ll find the tools you need at the Eclipse Web site (see the list of resources at the end of this article). You’ll need:

- The base Eclipse 3.2 platform
- Java support (JDT)
- C++ support (CDT) if you plan to analyze C/C++ code
- TPTP 4.2 platform or later (you’ll use a TPTP 4.3 build for this article)

If you choose to use Callisto, you can select and install almost everything you need, including the Java development tools; however, if you plan to analyze C or C++ code, you need to download the TPTP C++ Code Review components from the Eclipse TPTP Web site—at this point, these components are still in a technical preview state.

Once you have an environment set up and Eclipse is started, you’ll notice a couple of new user interface elements. TPTP static analysis adds new toolbar items to the Eclipse Workbench, and you’ll also see some new options available from the Eclipse Run… menu.

To use the static analysis tool, you must create one or more analysis configurations, which are simply lists of rules you want to apply to your source code. You can have as many configurations as you like and can use them from many places in the Eclipse user interface, such as the main toolbar and menu, the Package Explorer or even within an editor.

Building an analysis configuration is easy. From the Run… menu, select the Analysis… option, and the main configuration dialog will appear. This window closely resembles the configuration dialogs for running and debugging applications in Eclipse. Select the Analysis entry in the left side of the window and create a new configuration using either the Toolbar button or context menu in the dialog.

Once a new configuration has been created, the right side of the dialog will display a tab folder with two tabs. The first, the Scope tab, is used to determine the default range of resources over which analysis will occur. Note that the values set in this tab are only a guideline and can be overridden by the user depending on the location where static analysis is invoked in the Eclipse user interface. The scope selected in this tab is used only when the user starts static analysis from the main menu or toolbar. For now, leave the scope selection at its default value: the entire workspace.

The second tab in the static analysis configuration dialog, the Rules tab, provides functionality for selecting different kinds of analysis and rules. The rule list is presented as a tree where the top-level nodes represent the analysis provider (for example, types of analysis) and the deepest nodes represent rules. All the nodes in between are rule categories, which are simply groupings of rules and categories that match a particular characteristic. Click the Code Review for Java analysis provider checkbox to select all rules for Java code.

When the rules and scope are configured, click the Analyze button to begin the code review process. If not already visible, an Analysis Results view will appear, displaying a historical list of each analysis that has been performed.

Congratulations! Your first automatic code review is now complete, and you can browse the results that have been reported, view highlighted problems in a Java editor, or, in many cases, perform an automatic quick fix.

One other aspect of TPTP’s static analysis feature is the ability to run multiple forms of analysis at the same time. This capacity is unique in the static analysis mar-
Static Analysis

ketplace. Most commercial vendors would be happy to sell you a different tool for Java code review or metrics, or similar tools for C++ or other languages.

Aside from the multiplied costs of owning several different tools, usability issues often arise, since each tool probably functions differently. With TPTP, you can simply select multiple analysis providers when the configuration is created, and a single analysis process can test code for both languages.

WHAT’S IT GOOD FOR?

Now that you’ve got a list of static analysis results, what should you do with it? Obviously, you’ll want to fix any problems you find, but first, you need to understand what the code review is reporting. Let’s look at a simple code example. Suppose you have a method like the one shown here:

```java
public String[] getList(boolean flag) {
    if (flag) {
        String[] list = new String[3];
        list[0] = "one";
        list[1] = "two";
        list[2] = "three";
        return list;
    }
    return null;
}
```

When you run Java code review on a class containing this example, you’ll get the following analysis result for the method:

Avoid returning null instead of empty array

So what’s the problem? Well, assume this method is used as it is in the following fragment:

```java
for (int i = 0; i < getList(flag).length; i++) {
    // Do stuff
}
```

Now do you see the problem? What happens if getList() returns null? The for loop will generate a NullPointerException, and the application will fail. If the flag that controls the method is invoked by a rarely used feature in your product, a basic JUnit test may not find this problem during development.

While this example is trivial, it highlights a common coding practice that often goes unchecked: The subtle problems that could be found easily by a basic automated code review typically evade detection until your product has reached a customer. With TPTP’s static analysis tools, you can find this sort of bug in seconds, early in the development cycle when it costs almost nothing to fix.

TPTP currently supports more than 70 Java code review rules (43 C++ rules) to detect many common programming problems. Of course, not every problem reported is serious. You’ll find code review results divided into one of three severities (Severe, Warning, Recommendation), but regardless of severity, the issues reported are worthy of investigation and correction. So why would you not run TPTP’s code review? It’s easy to set up and can quickly examine even large workspaces to find problems that developers commonly create. Best of all, you can download code review for no cost.

DEVELOPING FOR STATIC ANALYSIS

Eventually, the rule sets available with TPTP will show their limits. For example, suppose you have an internal API for which enforcement rules are needed. Many of the other free and open source alternatives to TPTP offer no extensibility or restrictive, confusing APIs. Most commercial analysis tools provide rich APIs, but these products can be cost-prohibitive. TPTP offers a better solution because its code review framework is both cost-effective and simple to extend. New rules can be created from existing templates or can be constructed as Java classes within an Eclipse plug-in. In either case, TPTP offers you the opportunity to use the power of the Eclipse editing and debugging facilities to create new rules.

TPTP’s static analysis framework offers a number of key extension points; however, when writing new rules, only a few of these are needed. The following table lists all of the extension points defined for static analysis use—note that all extension points are in the org.eclipse.tptp.platform.analysis.core namespace.

<table>
<thead>
<tr>
<th>Extension Point</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>analysisCategory</code></td>
<td>Defines all of the categories used in TPTP</td>
</tr>
</tbody>
</table>

SETTING UP A PLUG-IN FOR NEW RULES AND CATEGORIES

The confines of this article prevent a detailed description of rule plug-in set-up; however, you can find a detailed tutorial on the TPTP Web site (see the resources at the end of this article). To define this process in its simplest form, take the following steps:

1. Create a new plug-in project in your Eclipse workspace.
2. Add the following plug-in dependencies to the project:
   a. org.eclipse.core.runtime
   b. org.eclipse.jdt.core
   c. org.eclipse.tptp.platform.analysis.core
   d. org.eclipse.tptp.platform.analysis.codereview.java
3. Define your new categories and rules.

Creating New Categories. If you have the source code for TPTP’s Java code review rules installed, open the plugin.xml file and you’ll see the analysisCategory extensions that define all of the categories used in TPTP.
Let's take a closer look. First, notice that the class field of each category points to the DefaultAnalysis Category class, which is supplied by the TPTP static analysis framework to provide a complete and commonly used class for categories. While it's possible to create your own category class, the default behavior should be the preferred way to supply this functionality.

Category extensions also declare an id field that should contain a unique identifier for the category. Colliding with identifiers defined by other categories usually produces undesirable effects since the framework will be unable to distinguish one category from another.

One final subtlety of the category extensions is the use of the provider and category fields. These mutually exclusive fields define where in the analysis hierarchy the category will be situated. If the provider field is used, the category is defined as a top-level category, while defining the category field will cause the new category to be contained within another category. In either case, the value of these fields is the unique identifier of the owning provider or category, respectively. Note that your new category can be placed within an existing category defined by the TPTP’s Java code review simply by using the correct identifier (you can find these identifiers in the plugin.xml of the Java rules plug-in).

**Creating New Rules.** Defining your own rules is a two-step process. First, create a rule class. Then, define an extension to map the rule class into the TPTP static analysis space.

While a detailed description of rule creation is beyond the scope of this article, a high-level view can be offered to help convey the concepts you will encounter. For a more detailed discussion of Java code review rules, refer to the tutorials on the TPTP Web site.

When you’re creating rule classes, the TPTP static analysis framework goes a long way toward simplifying the task for you. If you always extend the AbstractAnalysisRule class, your job is half finished. This abstract class provides all of the common functionality needed by every rule, and all you really need to do is implement your own worker method to set the rule criteria.

For the example, here you’ll create a rule that generates results anytime the Thread.stop() method is used. In basic terms, you want to parse all the source code and look for classes that extend java.lang.Thread and call stop(). Listing 2 shows an implementation for this rule.

<table>
<thead>
<tr>
<th>Extension Point</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>analysisProvider</td>
<td>Provisions a new type of analysis into the TPTP static analysis framework. Analysis providers are the engines that control the parsing of source and execution of rules.</td>
</tr>
<tr>
<td>analysisCategory</td>
<td>Defines new analysis categories. These can be owned by an analysis provider or by other categories.</td>
</tr>
<tr>
<td>analysisRule</td>
<td>Defines a new analysis rule. Rules are the workhorses of static analysis, responsible for executing on defined criteria and generating results when those criteria are met (or not met).</td>
</tr>
<tr>
<td>analysisRuleQuickFix</td>
<td>Defines a process used to correct the results of rules. Quick fixes usually encapsulate a refactoring class that manipulates source code directly to eliminate the cause of a generated analysis result.</td>
</tr>
<tr>
<td>analysisRuleTemplate</td>
<td>Think of rule templates as unfinished rules. Templates can be used to create a skeleton for a repetitive rule and provide fields that can be filled in by either a developer or end user to create new rules.</td>
</tr>
<tr>
<td>analysisRuleSet</td>
<td>Rule sets are used to define a quick selection set of rules. For example, a rule set could easily be created that selects all rules defined in Joshua Block’s “Effective Java” (Addison-Wesley, 2001).</td>
</tr>
<tr>
<td>analysisViewer</td>
<td>Analysis viewers are used to examine the domain-specific results of an analysis. Viewers can take any form that fits the presentation of data (for example, tree, table, diagram).</td>
</tr>
<tr>
<td>analysisExport</td>
<td>An analysis exporter is used to define a format for exporting result data. Defined exporters appear as selectable user interface items when the end user chooses to export data from an analysis provider.</td>
</tr>
</tbody>
</table>
the analysis provider, as shown in the example rule.

At this point, you can take one of two paths to implement your rule. First, you can use any of the JDT APIs implemented in the Eclipse platform, or you can use the simplified API supplied by the TPTP analysis framework—the latter is streamlined for handling code review queries and generating results, so the AvoidThreadStop example will take this approach.

After determining the resource being analyzed, the code next acquires a list of method invocations from the resource, then calls the ASTHelper.satisfy() method. This method filters out data that doesn’t fit the filter criteria. In our example, the code uses the MIFILTER array to discard any method invocations for methods not defined in the java.lang.Thread class and to subsequently discard any methods that are not calls to stop(). What remains is a list of calls to Thread.stop().

Since this rule is trivial, our work is finished. In this case, any method invocations remaining in the list are considered violations of the rule, so the generateResultsForASTNodes() method is called to create results for each of them.

Finally, the Java class being reviewed could be one that extends java.lang.Thread, so you need to take one more step to check any super method invocations for the same Thread.stop() criteria and create results for those, as well.

With the rule class created, only one easy step remains. The rule needs to be integrated into the static analysis code-review structure so it is available and selectable by the user. To do this, create a simple extension in the plugin.xml as follows:

```xml
<extension
   point="org.eclipse.tptp.platform.analysis.core.analysisRule">
   <analysisRule
      category="codereview.java.category.threads"
      class="com.mycompany.rules.AvoidThreadStop"
      id="myrules.java.threads.stop"
      label="Avoid using Thread.stop()"
      severity="2"/>
</extension>
```

Notice that the category identifier for your rule specifies one that already exists in TPTP. This will plug the rule into the existing TPTP Threads category. When you start a runtime workbench containing your rule plug-in, the analysis launch configuration dialog should contain your new rule.

**IT'S ONLY THE BEGINNING**

In a short article like this one, it’s difficult to provide comprehensive details of a complex subject like static analysis; however, you’ve gone through some of the basic concepts of the TPTP framework. You should now have at least a cursory understanding of how to use the static analysis component from an end-user perspective, and if you’re a developer, you should have a footing in place to help create your own rules.

If you’re interested in a complete step-by-step procedure for creating your own rules, refer to the TPTP Web site, where you’ll find several articles discussing rules as well as more advanced topics not covered here.

Keep in mind that TPTP is an open source project to help the entire community, so if you develop some interesting rules or integrate a new type of static analysis provider, by all means submit them to the project so that everyone can benefit.

**RESOURCES**

- Eclipse www.eclipse.org
- TPTP www.eclipse.org/tptp
Listing 2: AVOIDTHREADSTOP CLASSES

```java
public class AvoidThreadStop
    extends AbstractAnalysisRule
{
    private static final IRuleFilter[] MIFILTER = {
        new DeclaringClassRuleFilter("java.lang.Thread", true),
        new MethodNameRuleFilter("stop", true)
    };

    /**
     * Analyze this rule
     *
     * @param history A reference to the history record for this analysis
     */
    public void analyze(AnalysisHistory history) {
        // Extract the resource being analyzed
        CodeReviewResource resource = (CodeReviewResource) getProvider()
            .getProperty(history.getHistoryId(), CodeReviewProvider.RESOURCE_PROPERTY);

        // Find matched in method invocations
        List list = resource.getNodeList(resource.getResourceCompUnit(),
            ASTNode.METHOD_INVOCATION);
        ASTHelper.satisfy(list, MIFILTER);
        resource.generateResultsForASTNodes(this, history.getHistoryId(), list);

        // Find matched in super method invocations
        list = resource.getNodeList(resource.getResourceCompUnit(),
            ASTNode.SUPER_METHOD_INVOCATION);
        ASTHelper.satisfy(list, MIFILTER);
        resource.generateResultsForASTNodes(this, history.getHistoryId(), list);
    }
}
```
We’re headed toward an era of dynamic rich Internet applications for the enterprise. Even in the face of heated debate about these types of applications and exactly where companies are on the adoption curve, it’s clear that more and more enterprises are building them.

I believe that rich Internet applications are superior to standard Web applications. As human beings, we tend to buy, use or pursue the “better” in daily life—and Web applications are no different. Consumers and enterprises will want to use rich Internet applications, and businesses must offer such applications, simply because they are superior—and if you don’t offer them, someone else will.

You can build rich Internet applications with Asynchronous JavaScript and XML, and you can build AJAX applications with JavaServer Faces. JSF is a standard, user interface component–based framework for building Java-based Web applications—with the emphasis on component-based.

The component approach is what gives JSF its power and allows exploitation of AJAX. With JSF, you don’t think in terms of markup, like HTML, any more. Rather, the paradigm is development with user interface components.

**COMPONENT CONSCIOUSNESS**

A JSF user-interface component has renderers for component markup. These are simply Java classes that know how to generate all the necessary markup (read: HTML) for a UI component. Just keep in mind that JSF is all about creating powerful UIs. Java is excellent for the back end, but it was always missing something on the UI side—and that’s exactly what JSF provides.

For application developers, it’s natural to think in terms of user interface components. A component can be as simple as the familiar input text field to a sophisticated component such as a tree or a tabbed panel. As a developer, you design the application from your reusable user interface components—if you use Swing or ASP.NET, you should find this approach familiar. Both technologies utilize components to create user interfaces. In fact, I like to think of JSF as bringing Swing-style development to the Web—building Web applications out of UI components.

If you’re not sure what I’m talking about, I’ll approach it a bit differently. In the JSP world—to print a collection of records of some sort, you put `<table>..</table>` HTML tags, create some sort of loop in Java, and put `<tr>..<td>..</td>..<tr>` tags inside the loop. Inside, you could have if statements to do something extra. This is the markup approach; it works, but with JSF this is done a lot simpler. You take a ready-to-use component, like dataTable, and bind it to the same collection of records. The component will then render all of the necessary markup and data for you.

**A WIDER VIEW**

Let’s zoom out and look at this at a broader level. If you’ve ever seen an assembly line for cars on TV, you would see a door, for example, being attached to a car. Now, think of the door as a ready-to-use component that’s being attached to car. You don’t see engineers building the door on the assembly line. Similarly, with JSF, you create Web applications out of reusable JSF components. There’s no need for an application developer to create the table component just before he uses it in an application.

You can find hundreds of JSF resources on the Web. To get started or learn more, try www.jsfcentral.com, an excellent site with a lot of news and resources. Another site, www.jsftutorials.net, can help you get started. To work with JSF, all you need is Notepad or any other text-based editor, though there are commercial tools that can help make it easier.

But all this is less important than the plain fact that JSF
is the only standard Web component technology for Java. By contrast, JavaServer Pages, portlets and servlets are standard, but aren’t component-based, and the Struts framework is neither component-based nor standard. Did we mention huge industry support? Certainly nobody wants to stay with a big, important application based on a funky open-source framework that was invented and developed by one clever programmer! This is why IBM, Oracle, Sun and others are cultivating JSF, carefully selecting the best ideas for Web development and paying big money to the best architects and developers to improve JSF for the rest of us.

JSF’s position as a standard is doubly important because it allows JSF to fully realize its promise as a component-based framework. The chances that components from different vendors will coexist in the same application are much better when the technology is more standardized—a great boon for all enterprise applications developers!

Another interesting facet of JSF is that it’s a framework for both application developers and component developers. Applications developers build Web applications out of JSF user interface components, while component developers build such components with JSF.

RICH INTERNET APPLICATIONS

Look at Google Maps. The application feels a little different than what we’re used to on the Web with the submit-and-wait experience. We get to a page, enter some information and click submit. Then, we wait for a reply. It’s pretty much unchanged since 1994, the early days of the Web. But let’s go back to Google Maps. The application feels fast; we don’t need to wait for it to reload or refresh. It almost feels like we’re running a desktop application.

Google introduced Google Maps as one of the first truly rich applications, laying the groundwork for the rise of AJAX as a technique for creating rich applications. AJAX allows us to refresh or update only a part of a page, instead of submitting a whole page every time. That’s why the applications feel fast, rich and interactive. With AJAX, you can update only the small part of the page that actually needs to be updated, but don’t update the parts that stay the same.

AJAX itself is not a framework, but a combination of existing technologies such as DHTML, JavaScript, XMLHttpRequest and DOM (Document Object Model). You can find numerous AJAX resources on the Web that will show and explain the underlying technologies in more detail.

AJAX lets you create rich Internet applications. However, it’s not the easiest technology to use. AJAX requires a deep understanding of the underlying technologies, such as JavaScript. In other words, creating sophisticated Web applications with AJAX technology is challenging and requires experts. It’s not uncommon to spend 80–90 percent of development time battling JavaScript incompatibilities among Web browsers and platforms.

COMBINING JSF WITH AJAX

We know that JSF is a good component-based technology for building enterprise Web applications. We also now know that AJAX lets us create rich, fast and interactive Web applications. What if you combine the two technologies to add some “richness” to JSF? If you use the two technologies together, take a very powerful user interface component frame and add AJAX functionality, you can build rich enterprise Internet applications.

Above, I said that AJAX could get complicated pretty fast. That’s still correct. However, because of JSF’s component architecture, you can easily use AJAX. How? You create JSF components that render all the necessary JavaScript, HTML and XML. You then simply use JSF AJAX-enabled components as out-of-the-box building blocks. This way, you can build rich Internet applications with JSF and AJAX without writing a single line of JavaScript—all the complexity of AJAX is hidden.

This is exactly what I meant by saying that JSF is the perfect match for AJAX. A simple JSF component such as an input field knows how to render its entire required coding. In the same way, a JSF component can send an AJAX request to the server after some event (such as onkeyup) and then update only a part of a page after a response is sent back. In this case, the component will also know everything about how to render all the necessary JavaScript and HTML for itself.

The point that I’m trying to make here is this: JSF components render markup. For an application developer, it’s not important what’s rendered, simple HTML or complicated JavaScript. The development style remains the same: building Web applications out of JSF components. Using AJAX to make applications interactive shouldn’t change your development approach or jump outside the JSF component model; you simply continue building Web applications out of reusable JSF components.

AJAX4JSF LETS YOU SEND AN AJAX REQUEST AND SPECIFY WHICH AREAS OF THE PAGE SHOULD BE UPDATED.
IMPLEMENTING THE IDEA

Now get a little bit more concrete and see what match-making tools are available to put JSF and AJAX in sync. Ajax4jsf (http://ajax4jsf.dev.java.net) is an open source framework that does just that. It brings AJAX functionality to JSF in the form of a JSF component library. With this framework, using just JSF components can add AJAX features to an existing or new JSF application. Because Ajax4jsf provides standard JSF components, it’s a natural extension to JSF. The framework is based on JSF standards, provides an open architecture and allows mixing and matching with any other JSF component library, free or commercial.

The best way to show any technology is through an easy example, and that’s exactly what you’re going to do here. You’ll take a very simple JSF application and add AJAX functionality to it.

A SAMPLE APPLICATION

Before you start, plan what you’re going to do. Imagine a registration page on the Internet with 10 or more fields. You start entering all the information. When done, you click submit (remember, submit-and-wait). You get the same page back and realize that four of the fields have incorrect information. Whoops, So, you correct and resubmit the page. Wouldn’t it be much better if you could get some feedback as you typed in the values—some kind of live validation, similar to desktop applications? That’s exactly what you’re going to do with your application—you’ll add live validation. As you type, you’ll get instant feedback about the correctness of the value.

APPLICATION BEFORE AJAX

Now it’s time to check out a sample application:


2. Open the project in your favorite IDE, and you’ll see a standard JSF page, with one input field with an attached validator. If the value entered is shorter than three characters long, an error message will be displayed. Whoops, So, you correct and resubmit the page. Wouldn’t it be much better if you could get some feedback as you typed in the values—some kind of live validation, similar to desktop applications? That’s exactly what you’re going to do with your application—you’ll add live validation. As you type, you’ll get instant feedback about the correctness of the value.

ADDING AJAX

To improve this, the moment you’re typing and see that the value is incorrect, you want an error message shown without having to click Submit. Use the Ajax4jsf component library to add this AJAX feature to your application. Here’s what needs to be done.

1. Download the Ajax4jsf framework.
2. Copy the files, ajax4jsf.jar and oscache-2.2.jar, into the WEB-INF/lib folder of your application.
3. Register Ajax4jsf in the web.xml file:

   <filter>
     <display-name>Ajax4jsf Filter</display-name>
     <filter-name>ajax4jsf</filter-name>
     <filter-class>org.ajax4jsf.Filter</filter-class>
   </filter>

   <filter-mapping>
     <dispatcher>REQUEST</dispatcher>
     <dispatcher>FORWARD</dispatcher>
     <dispatcher>INCLUDE</dispatcher>
   </filter-mapping>

That’s it.

In your actual application, you need to make three small changes. (You can also just download the finished application here: http://webdownload.exadel.com/misc/eclipsereview/a4j-validation-complete.zip.)

1. Add a taglib declaration for Ajax4jsf, a standard step for adding any new custom tag library.
2. Add an <a4j:support event="onkeyup" reRender=""msg"">component inside <h:inputText> as a child element. This enables the sending of an AJAX request to the server as the user types. In other words, each time a letter is typed, an AJAX request is sent to the server. (You can modify the frequency of updates.) The request goes through the standard JSF page life cycle. This is not a client error message; you’ve invoked the standard JSF validator in this example. The reRender attribute will point to an AJAX area (see next step) that you need to re-render (update) as you send an AJAX request.
3. Enclose an <h:message/> component inside an <a4j:outputPanel id="msg">component to make it part of an AJAX area. You need it to be in an AJAX area because you want the dynamic error message to be displayed or cleared from just this area without having to hit Submit and update the whole page.

That’s all you need to do.

In this example, you’ve added AJAX functionality by simply using JSF components. You didn’t change your development approach of just using JSF components; the result is still just a standard JSF page. You didn’t have to make any changes to any other parts of the application, like the configuration file or the managed bean. You invoked the standard JSF page life cycle to validate input. That’s how simple it is.

How does it work? Ajax4jsf lets you send an AJAX request and then specify which components (areas) on the page to update. Keep in mind that you can update multiple components (areas) at once.

For example, consider a suggestion-box component with AJAX. As you start typing into it, the component shows a suggested list of values. You can place any number of suggestion-box components on a page, but none of them are able to update any other components on a page. So basically, all the AJAX functionally is “enclosed” within the component. That’s a component-level approach. However, for sophisticated applications, you must be able to specify which components (areas) need to be updated based on some events—page-wide AJAX support—and that’s exactly what Ajax4jsf provides.

Don’t get me wrong: The component-centric approach can still be used when appropriate. In fact, combining the two approaches will give you the ability to create truly powerful rich Internet applications with JSF and AJAX.
Company Buster—Or Company Booster?

If you talk to some people, you’ll hear that Eclipse has ruined the commercial development tools market. By this way of thinking, a free product has ruined the business prospects of development tools companies. Many people recount this nostalgically, pining for the days of dozens of tools companies; to their minds, Eclipse is a scourge in the marketplace.

On the other hand, a vastly more positive perspective is possible. By this reading, the existing tools companies were all exhausted, unprofitable hulks that lived in daily fear of the monster from Redmond. The release of the Eclipse framework revivified the existing non-Microsoft players, allowing them to emerge in a new form with higher long-term potential.

As one might expect, my viewpoint leans toward the latter interpretation. The tools business has always been a difficult one. Microsoft has done an excellent job in creating a tools/applications/server ecosystem for which their solutions are always the best-tuned for the environment.

With a significant portion of the overall market residing on Microsoft’s plate, that left smaller portions for the rest of the tools industry. Given the widespread availability of free tools, selling development tools is a rough job.

While it’s tempting to think that people will prefer commercial products to “inferior” free ones, my experience is different. I was once consulting to a tools company that provided a high-end, extremely capable product. Their tool made programmers more productive, no question about it—but they weren’t getting the sales they wanted. So I interviewed a few end users to understand the tool decision-making process, and, boy, were my eyes opened about the Power of Free.

One woman I spoke with about Java development efforts told me that the company didn’t have the budget to purchase fancy tools. I therefore asked her what they did use to do their Java development. I expected to hear something like emacs or even vi. Her response: Notepad. (I guess I’d have to put that organization into the Microsoft camp—although one wonders about the quality of code her organization turned out.)

So, for organizations like that one, Eclipse presents a real opportunity to move to state-of-the-art tools within a constrained budget. They can achieve higher productivity, faster time-to-market, better ability to recruit and retain skilled employees, and a more satisfied workforce. (Nobody could be happy programming Java in Notepad!)

Just as important, however, is the tool-developer side of the equation. While it’s tempting to paint Eclipse as a company buster, I believe it’s exactly the opposite—it’s a company booster.

ECLIPSE OFFERS THE OOMPH

Just as a booster rocket supplies initial thrust for a rocket-borne payload, Eclipse provides a tremendous amount of oomph for tools. Instead of having to create an entire framework within which to deliver a special functionality, a tools company can rely on Eclipse to take care of a lot of the heavy lifting—or perhaps heavy lift-off would be the right metaphor.

With the many projects in the Eclipse Foundation providing the technological underpinnings, for-profit tools companies can operate much more leanly and in tighter focus on their special area of expertise. Rather than having to fund, design, build and maintain a bunch of functionality that does nothing to differentiate their “secret sauce,” tools providers can deliver a highly concentrated functionality payload that serves them in the market.

Far from ruining the commercial tools market, Eclipse has strengthened it. If you take a look at the plug-ins on the Eclipse site, the variety and specialized capability available are simply amazing. It’s clear that Eclipse is accomplishing its job—providing a framework to enable differentiated development functionality to find its way into the hands of developers throughout the world.

---

Bernard Golden is Chief Executive Officer of Navica (www.navicasoft.com), a Silicon Valley consulting firm offering open source strategy, implementation and training services.
I thought application security was someone else’s problem.

Until last night’s SQL injection attack.
Join Us In 2007!

eclipse WORLD
The Enterprise Development Conference
Hyatt Regency Reston
Reston, VA

November 6-8, 2007

Washington, D.C. Area!

www.eclipseworld.net