EMBARCADERO
TECHNOLOGIES®

White Paper

Streamlining Offshore Java™ Development

Using JBuilder® to Overcome the Big Four Offshore Development Challenges: Application Quality, Application Performance, Distributed Team Collaboration, and Productivity

December 2008

Corporate Headquarters
100 California Street, 12th Floor
San Francisco, California 94111

EMEA Headquarters
York House
18 York Road
Maidenhead, Berkshire
SL6 1SF, United Kingdom

Asia-Pacific Headquarters
L7. 313 La Trobe Street
Melbourne VIC 3000
Australia

INTRODUCTION

Software development work is increasingly being off-shored or outsourced as companies try to reduce costs in efforts to remain competitive. However, these solutions have issues and challenges that can ultimately end up costing a company more money in the long run. This paper discusses how JBuilder® can help organizations who are in the process of off-shoring, or have already off-shored some or all of their development efforts as a means of reducing development costs.

We should say up front that while the terms offshoring and out-sourcing are often used interchangeably, there are important differences in the two models when it comes to software development. Outsourcing involves hiring an outside agency or company to do a development project; while off-shoring ensures that the development team is employed directly for whom the work is done. Yet, while the models are different, the issues and challenges for software development teams engaged in these models are very similar. Moreover, general trends of the software industry compound the problems already evident in both the offshore and outsource models.

Offshoring is similar to remote development and presents many of the same challenges. The remote locations where off-shoring occurs are often in regions of the world where labor costs are significantly less than higher-priced areas such as the United States, Japan, or Western Europe. Because of this, off-shoring adds the additional challenges of extreme time zone differences (e.g. 8+ hours), cultural differences in the workplace, and language differences. All of these have a significant impact on communication and productivity, and can ultimately affect the quality of the software being developed.

Within a few short pages, we'll talk about the challenges associated with these models, how the goals of your projects remain the same, what you can do to ensure success, and how JBuilder can help.

CHALLENGES WITH OPEN SOURCE

As offshore projects and developers get further from a central point of management, there is a greater need to have visibility into what open source software is being used and some control over how, or even if it should be used.

Although the adoption of open source software can have a profound, positive impact on software projects, there are also a few undesirable side effects that can be quickly compounded with off-shored or outsourced projects.

- Inconsistent and Unmanaged Developer Environments
- Licensing Issues
- Lack of Support

INCONSISTENT AND UNMANAGED DEVELOPER ENVIRONMENTS

With the advent of open source IDEs like Eclipse (http://www.eclipse.org) and NetBeans (http://www.netbeans.org) has come a growing market for plug-in providers and their solutions. There is already 1100 plug-ins available for Eclipse alone.

So, the question isn't really if developers will use plug-ins or not, but how will they choose which plug-ins to use, and how will the organization manage the use of plug-ins across the development team so they can circumvent any problems that arise.

For example, every developer who downloads a plug-in for their IDE has effectively created their own version of that IDE. Most likely, that specific plug-in hasn't been tested with all of the other components in the IDE, or with any of the other plug-ins that the developer might download.

In addition, there can be problems when a developer uses a plug-in or even a version of a plug-in that the other developers on the team may not have. It can be difficult, if not impossible, for someone else to enhance or modify the code created by the first developer if the plug-in leaves behind artifacts in the code (sometimes called marker code) that only the plug-in understands. This is common, for instance, in many visual designers for things like user interfaces (UI's), UML modeling, Enterprise JavaBeans™ designers, Web service designers, and so on.

Equally important are the licensing and support implications when they do this.

FRAMEWORKS, FRAMEWORKS EVERYWHERE

The number of open source frameworks and technologies has grown almost as fast as the number of plug-ins. In all, Wikipedia lists over 150 different Web application frameworks which developers can choose from that reportedly make them more productive and simplify application development.

Frameworks such as Apache Struts, Apache Open for Business (OFBiz), AppFuse, Spring MVC, and Tapestry act as blueprints or guidelines on how to build applications of a specific nature. Open source technologies such as Hibernate, JPA, CMP, and Reactor exist for Object-Relational Mapping (ORM) and persistence frameworks. And still others for security frameworks, template frameworks, caching frameworks, and testing frameworks.

In some sense the developer's time has been shifted away from writing code to evaluating, understanding, and selecting which of these 150 frameworks they should use as a basis for their application.

LICENSING ISSUES

There are almost as many different open source licenses as there are open source projects these days. The much-watched case *Jacobsen v. Katzer*, No. 2008-1001 (Fed. Cir. Aug. 13, 2008) established enforceability of open source licensing agreements and a recent article in TechNewsWorld gave this warning:

"It is important for companies to be aware of the implications of this decision and to respond accordingly; this applies to all companies that use open source software -- even those who think they don't. The temptation to incorporate open source software into a company's products is great, because open source software is readily available via download and is free of charge."

The article goes on to suggest regular audits, an adequate record-keeping system, and even purchasing insurance such as Open Source Risk Management ("OSRM"),

Companies need to proactively develop policies and procedures regarding the use of open source software, as the financial and legal risks associated with the improper use of open source software have been made clear by the *Jacobsen* case.

SUPPORT —YOU GET WHAT YOU PAY FOR

Even if you manage to track all of your open source code and ensure the licensing is taken care of, you still have to worry about having adequate support.

An IDC study referenced in a July 2008 article in Business Innovation stated that although cost savings remain the number one reason for open source adoption, organizations' spending on quality assurance, testing, and certification of open source systems would increase 150% between 2007 and 2008.

These cost increases make sense when you consider the following traits of open source code:

- Open source code is, by definition, almost never created with the intent or forethought of being turned into a commercial product.
- Frequently there are no or very few coding standards that govern the development of the code.
- Little or no user documentation is created, and documentation about the code itself is usually inadequate.
- There is no single owner (either individual or entity) of the code to whom you can go for help, information, or bug fixes.
- Releases, patches, and bug fixes are driven by the community or project members, and not by specific business needs or customer demands.

The Business Innovation article concluded, "IDC's survey identified two key inhibitors to the adoption of open source: the risk of copyright or patent infringement and the lack of availability of support."

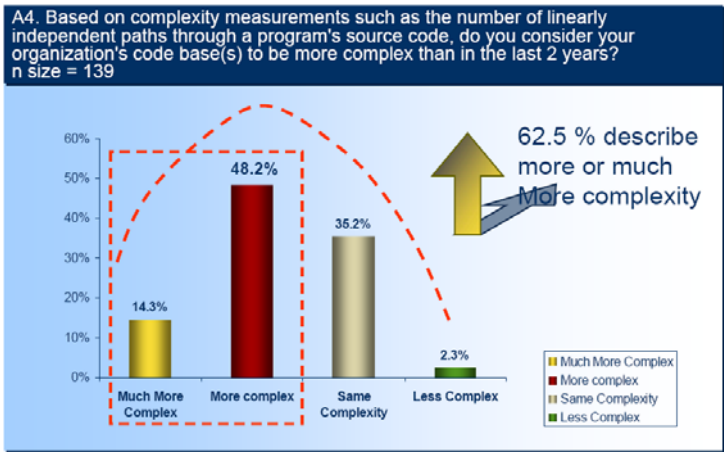
Growing Complexity of Applications

Developers are expected to keep up with not only all the advances in technology and the influences of open source, but also all of these business needs while meeting ever tightening schedules and budget constraints. Now, add the off-shoring factor on top of all this and the level of complexity grows even more.

As a case in point, in April 2008 IDC published a survey titled, "Debugging and Business Value Survey" in which 139 companies responded to the questions about the complexity and quality of the applications they are developing today.

When asked the question: Based on complexity measurements such as the number of linearly independent paths through a program's source code, do you consider your organization's code base(s) to be more complex than in the last 2 years?

An overwhelming 62.5% responded that they would describe their current code base(s) as more or much more complex than in the past 2 years.



The bottom line is that developers need better tools to help them to create more reliable, higher performing, and more complex applications. And when offshore and outsourced teams are added to the equation, these tools are an absolute necessity.

How to Get it All Done

In short, the goal of most IT managers, Line of Business (LOB) owners, or customers is simple - deliver the software application that meets my needs and expectations (which, by the way, may or may not the same as the formal, documented requirements) on time and within budget.

What makes this different in today's world isn't so much the "what', but the "how".

Companies will strive to achieve faster time to market by utilizing both outsourcing and off-shoring. With this, it is important to focus on a few key areas which have the largest impact on the development teams' ability to achieve these goals. Most of these areas shouldn't surprise or confound anyone, but what is new is how we address these challenges in the context of the new models for software development – namely outsourcing and off-shoring. Specifically, these areas are:

- Increase Developer and Team Productivity
- Improve application performance
- Meet or exceed software quality objectives
- Enable Global Development Teams

INCREASE DEVELOPER AND TEAM PRODUCTIVITY

Developer productivity is the whole point of an IDE – to seamlessly integrate editing, file management, compilation, debugging and execution into one, cohesive environment. Over time this has been extended to include more complete support for the full application lifecycle (e.g. requirements, design, testing, deployment, support, etc.), but it could be argued that this is more breadth of functionality than depth in any one area.

CODE BASE KNOWLEDGE

The question to ask is, "What is the key to team and individual developer productivity in maintaining and extending a large application"? Let's start by making the following assertions:

- A developer's knowledge of an application code-base is likely the single biggest factor of individual productivity.
- Correspondingly, the team's collective knowledge of an application code-base is the single biggest factor of team productivity.

When you think about an individual's or a team's productivity or lack thereof, you will likely come up with a number of answers: knowledge of API's, libraries, frameworks, and programming language, knowledge of the application domain, effective tools, development processes, continuous builds, unit tests, and so on. But, above all of this, it only stands to reason that an individual's productivity is directly proportional to his knowledge of the code-base and the team's productivity is directly proportional to the team's collective knowledge of the code-base.

REDUCING APPLICATION COMPLEXITY

Developer productivity can also be dramatically impacted when the complexity of application development is reduced.

For example, once the decision is made to use Struts as the framework for a Web application, it is reasonable to assume that an IDE can be Struts-aware and actually incorporate features that make using Struts and creating a Struts-based application much easier for the developer. Having Struts support in the IDE can help reduce the complexity of using that framework and make it easier for the developer to successfully build a high quality Struts application in less time compared to using an IDE that was not Struts-aware.

Re-using Code

Although most would agree that code re-use has the potential to improve developer productivity, the reality is that achieving significant levels of code re-use has eluded most development teams. The problem typically isn't the lack of code to re-use, but the knowledge on *how* to re-use it. There are probably millions of lines of code samples, code snippets, and modules, as well as libraries and repositories full of components and services that are available on the Internet and even within the walls of an organization that are simply not being used, or more correctly, re-used.

Taking this one step further, if these concepts can be extended to the development team versus the individual developer, there is an even larger potential gain to be had.

Improve Application Performance

When application development is outsourced or off-shored, a development team has to get their arms around a body of code that they probably didn't create, quickly gather some performance data about that code, analyze the data, and determine what changes are required to meet the new performance goals required to meet the needs of the business. Moreover, if the development team doing the profiling and analysis of the application is located offshore and they need to collect data about an application that is running in IT facility at corporate offices, they need the ability to do remote profiling while at the same time analyze the data locally.

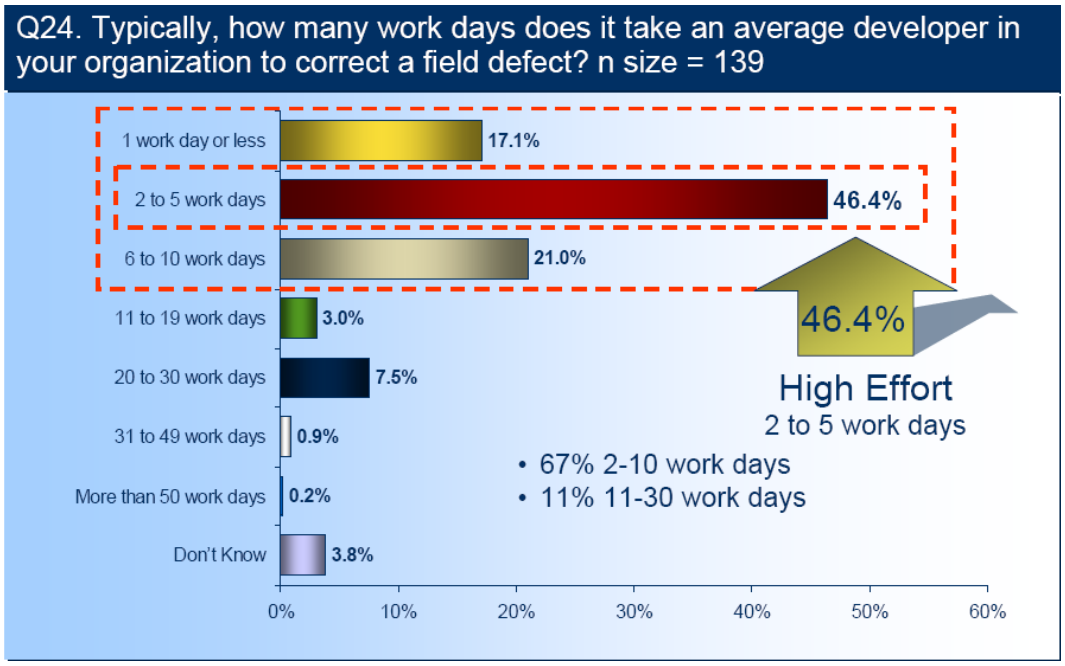
Documenting the improvement in the application's performance is even more challenging. Once the changes have been made to the code base and the application is redeployed, it may be necessary to show a before and after snapshot of the application's performance for purposes of comparison or documentation to management or executives that the performance goals have been meet.

Meet or Exceed Software Quality Objectives

There is a very real financial reason behind the drive for high quality software. In same IDC survey mentioned earlier, 67% of the 139 respondents indicated that it takes between 2 and 10 days for an average developer in their organization to correct a field defect.

Looking at an average case of 5 days and the typical cost of a developer in the US at a very conservative $60/hour, this would be $2400 for each field defect that had to be

addressed, which doesn't sound that bad. But that's just the tip of the iceberg. What if it takes 10 days and a senior developer is need, or a team of 2 or 3 developers to identify and correct the problem. Assuming $120/hour for a senior person, that cost could go up to nearly $30,000 ($120/hour, 8 hours/day, 10 days, 3 people).



Now it starts to get interesting. That's nearly the cost of two offshore developers for a full year. And if that isn't bad enough, imagine if the defect meant that customers couldn't place orders on your Web site for 5 or even 10 days, or you couldn't process or ship orders and recognize the revenue during the last few days of the quarter. It may take time and effort to build quality software, and when working with an offshore team it may even take a little extra time and effort to achieve the same levels of quality as when the development team is all sitting in the same building. The inescapable truth is that while quality may cost, the lack of quality costs even more.

Enable Global Development Teams

If setting up a development team offshore saves a project tens or hundreds of thousands of dollars, but the project is 6 months late due to lost productivity of having people disconnected from the core team, then what actual savings were realized? The trick is to create a distributed development team that works as effectively, efficiently, and cohesively as if they were all sitting in the same room.

Likewise, the ability to take advantage of additional resource pools in different locations and quickly add them to an existing project to make up lost time can be a major factor in improving time to market. To do this, you need a tool that enables maximum transfer of knowledge between developers and teams about the application being developed. You want a tool that makes replicating development environments, including the full

ALM (Application Lifecycle Management) stack, as quick, easy, and error-free as possible. You want tools that enable new members of the development team to be able to quickly learn about existing code by providing models which are derived from the existing code base.

You also want to make sure the developers can effectively communicate and collaborate with each other. As one site is closing down for the day and another one is starting up, it is important for each team to know what the other has done. The team just coming online will want to know what code had been changed by the previous team, what progress (or in some cases, regression) was made, and what new issues they uncovered.

Bas de Baar offers a list of suggestions to project managers on how to effectively manage offshore projects. Number seventeen on his list encourages managers to "Monitor progress in detail". He goes on to say that, "It is important that you track the important issues until they are sorted out completely". When thinking about what needs to been done to enable the global development team, it's important to think about what is needed by both the onshore and the offshore members of the team in order for the project to be successful.

JBUILDER HELPS YOU DELIVER

We've looked at some of the leading trends in the area of offshore software development, and also at some of the ongoing goals and expectations that organizations expect from their developers and the software they deliver. The question which naturally follows is, "now what"? Is there a way to keep up with the trends, or even take advantage of them, while at the same time meet the objectives of the business, both technically and financially? The short answer is "yes", but the harder question is how.

In his article, "10 ways to increase the productivity of your programmer", Justin James suggests, "Stop hammering nails with a screwdriver" as one way to improve the efficiency and productivity of your development teams. He goes on to say that, "Many of the tools that can help your team are not open source or freeware, for better or for worse. There are few tools on the market that cost more than a week's salary for a programmer, but there are many times when using the wrong tools or no tools wastes much more than a week."

Following are some specific examples of how JBuilder can help meet the issues and challenges faced by globally distributed development teams. In addition, there is some data extracted from a Cost Xpert study which demonstrates that not only can JBuilder meet these needs, but it can do it cost effectively and with a lower Total Cost of Ownership (TCO) than some of the leading competitors.

DEVELOPER PRODUCTIVITY

Using the UML™ modeling tools in JBuilder, the exchange of knowledge and information about existing source code between remote teams and individual team members can be greatly improved. For example, Class, Sequence, and Use-Case diagrams help developers get a big picture of the structure and flow of the code without having to dive directly into the source and try to figure this out for themselves. These diagrams, which can be derived from existing source code using JBuilder's LiveSource™ technology, can eliminate or reduce language barriers and other challenges which can impede team progress. As they say, a picture is worth a thousand words, or in this case it could be worth 100's of thousands of lines of code.

JBuilder also includes a number of visual design tools and modelers that can significantly increase developer productivity, and also improve code quality by reducing the chance of coding errors. Also, by using these designers and modeling tools, developers located in different locations can more effectively communicate and share information about pieces of the project they're working on since they can use the models to convey structure, logic, and content with each other rather than relying on each team member to pour over the code to gain this understanding.

JBuilder 2008 provides modeling support for Enterprise JavaBeans™ (EJB™) and Java Persistence API (JPA) projects. The EJB Modeler enables developers to create a visual model as they develop EJB applications. With JPA and modeling features, developers can create a Java™_ modeling project with JPA support. JPA projects can be based on standard persistence technologies such as Hibernate, TopLink, and others.

The Web Services Designer includes tools for Web services development. These tools assist the developer with the following aspects of Web services development:

- Discover. Browse UDDI repositories or WSIL documents to locate existing Web services.
- Create or Transform. Create bottom-up Web services from existing artifacts, such as JavaBeans and Enterprise JavaBeans (EJBs). Create top-down Web services from a WSDL discovered from others or created using the WSDL Editor.
- Build. Wrap existing artifacts as SOAP accessible services and describe them in WSDL. The Web services wizards assist in generating a Java client proxy to Web services described in WSDL and in generating JavaBeans™ skeletons from WSDL.
- Deploy. Deploy Web services into a variety of test environments.
- Test. Test Web services running locally or remotely in order to get instant feedback.
- Develop. Generate sample applications to assist in creating a Web service client application.
- Publish. Publish Web services to a UDDI v2 or v3 repository; advertise Web services so that others can access them.

The **Swing Designer** in JBuilder is a powerful and easy to use bi-directional Java GUI designer that makes it very easy to create Swing-based applications without spending a lot of time writing code to display simple forms.

TEAM DEVELOPMENT AND COLLABORATION

JBuilder brings advanced ALM support with ProjectAssist™ and TeamInsight™ which provide "best in class" integration of Open Source Software (OSS) ALM tools. ProjectAssist and TeamInsight are JBuilder features that install and facilitate the use of the components that make up the ALM stack. The goal of these tools is to help coordinate teamwork and thereby optimize your team's efforts. TeamInsight is a set of project tools that enable development teams to coordinate their work and to optimize their efforts. ProjectAssist provides the server install, configuration and assimilation of the ALM components by the ProjectAssist Administrator. As part of the ProjectAssist install, the Administrator defines projects and team members for the projects. The team members can then coordinate their efforts through the use of the various TeamInsight tools. Additionally, as new projects are started or existing projects are moved from one location to another, you will be able to get the development team up and running quickly using the ProjectAssist deployment capabilities. TeamInsight and ProjectAssist provide direct integration with Bugzilla, Continuum, CVS, Borland® StarTeam®, Subversion®, and XPlanner. Integration with additional systems is made possible by using the published APIs.

TeamInsight provides code, project and team management capabilities where individual developers have a unified, real-time view of their project responsibilities for bugs, change requests, code notes, tasks and requirements, and the entire team has a shared project Web portal with live data and statistics on team vector and velocity. TeamInsight enhances collaborative development with its centralized portal that allows team members to monitor project activity for the source code repository, track recent check-ins, view quality metrics, even view live burn-down charts for project progress. Project managers have instant reporting on the projects and know how a project is really doing.

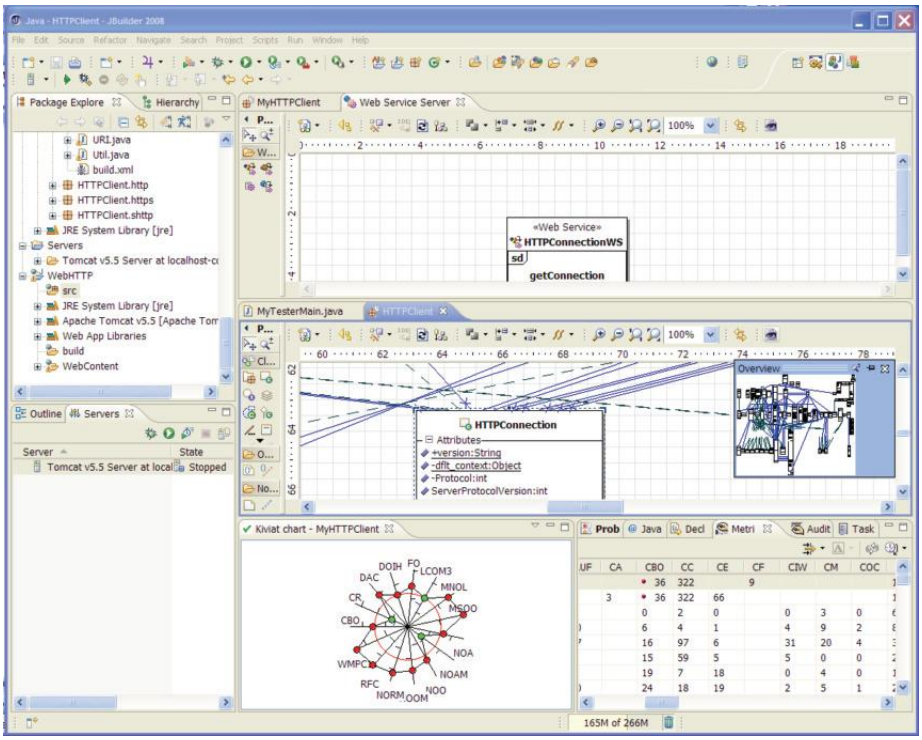
ProjectAssist provides simple, single-point installation and configuration of a complete developer tool stack for bug tracking, source code management, project planning/tracking, requirements management and continuous integration builds, dramatically reducing the time and complexity to create new team and project definitions. Wizards facilitate new installations, and even allow assimilation of existing installations for source code management, defect tracking, requirements management, project planning and more. ProjectAssist not only reduces the time it takes to install a developer stack, but can also reduce the potential of human error as a result of manual configuration and integration processes.

Communication and collaboration for teams through JBuilder's exclusive **peer-to-peer technology** enables teams to conduct code reviews, debug code, and discuss requirements within the shared programming environment. Communication is as easy as adding a workgroup and then selecting the desired person in that group to initiate communication. Peer-to-peer collaboration features allow two or more users to collaborate across a local area network (LAN) and send data. Peers are discovered automatically when they are on the same LAN.

Besides being a great productivity tool, the UML modeling capability in JBuilder fosters better communication between developers, developer teams, and management. JBuilder 2008 supports versions 1.4 and 2.0 UML specifications and includes **LiveSource®**, which gives the ability to keep code and diagrams in-synch at all times. Plus, with the ability to generate automatic Sequence diagrams and full documentation communications is enhanced for both the developer and the management teams.

Code Quality and Application Performance

Using the **static code analysis** capabilities in JBuilder, which include over 90 software metrics, developer teams can ascertain the overall quality of the project design. This allows them to find complexity, cohesion between objects, test coverage, and many other development aspects that help pinpoint maintenance and performance nightmares.



The product also includes over 200 software audits that help with the understanding of the type or style of the code being worked on. The static code analysis tools can help find design flaws, duplicate code, synchronization issues, and other issues to ensure

code being created is to the highest standards. Static code analysis tools ensure that even with multiple offshore teams working on a project, the overall quality of the code can be measured, monitored, and maintained to organizational standards.

The performance analysis tools in JBuilder include memory (resource consumption), CPU (time consumption) and Web (component and container activity) profiling. These abilities give the developers an easy to use set of tools that will allow them to understand performance issues in minutes, and down to the exact lines of code that are causing them. The product also includes an advanced thread debugger, which allows for predictive analysis of what and where threads may misbehave.

When code is developed over time by several developers, portions of code and sometimes whole methods are no longer used, but remain dormant in the code. This type of dead code makes an application longer and more difficult to understand. The code coverage tools in JBuilder make locating dead code easy so that you can clean-up and simplify your application. Clarity of code makes it easier to read and more reusable. Code coverage also makes it easy for you to identify how much of your code has been tested by highlighting all tested code. Understanding exactly which classes, methods, and lines of code have not been used allows you to modify your test plan to cover all areas of the code.

For offshore development, all of these capabilities are available via a remote agent which collects data about the running application and runtime environment and presents it to the developer running the analysis tools on a local or remote desktop.

CODE COMPREHENSION AND REUSE

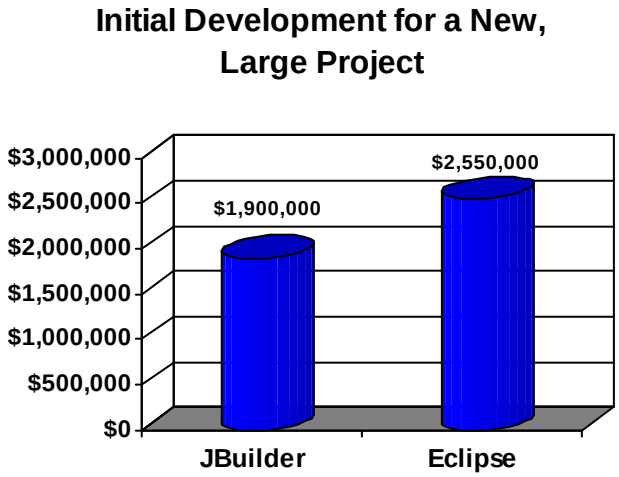
Application Factories introduce an application-driven development paradigm, where the structure, evolution, and logic behind the development of the application are captured and maintained as part of the development project within JBuilder. Application Factories allow for increased reuse of not just snippets of code, but entire application modules which encapsulate complete functionality (e.g. a function point, for instance), or even entire applications. The advanced tooling in JBuilder allows for increased navigation of code using *Tagging*, and has the ability to help add developer knowledge and intent to code with *Scripting* and *Templates* that make creating reusable functionality a reality. Metadata, which stays attached to the code when it is created, can be opened by any subsequent developer to understand the context and purpose of code snippets, methods, and classes which are part of the application.

HIGHER PRODUCTIVITY, LOWER COSTS

In an attempt to quantify the cost of productivity as it relates to the use of various Eclipse-based IDEs, Cost Xpert® (http://www.costxpert.com) conducted an extensive study which modeled small, medium, and large development teams doing both new development and maintenance on various size projects. The report compared the Total

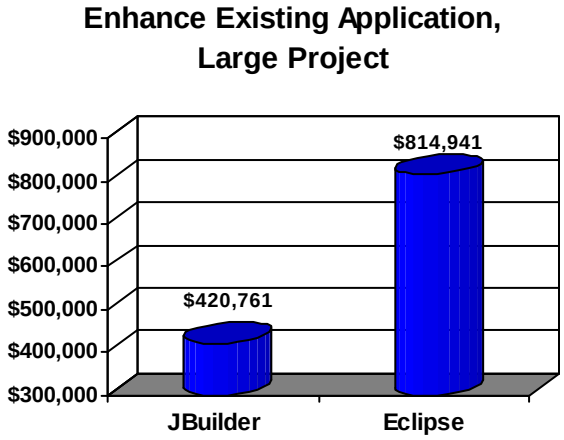
Cost of Ownership (TCO) of three popular commercial Java IDEs to that of the free Eclipse IDE.

In this study, team configurations and projects of varying sizes and purposes were modeled and measured using two scenarios: (1) building new Java software and (2) enhancing/maintaining existing Java applications.



Cost Comparison of JBuilder versus Eclipse for New Projects

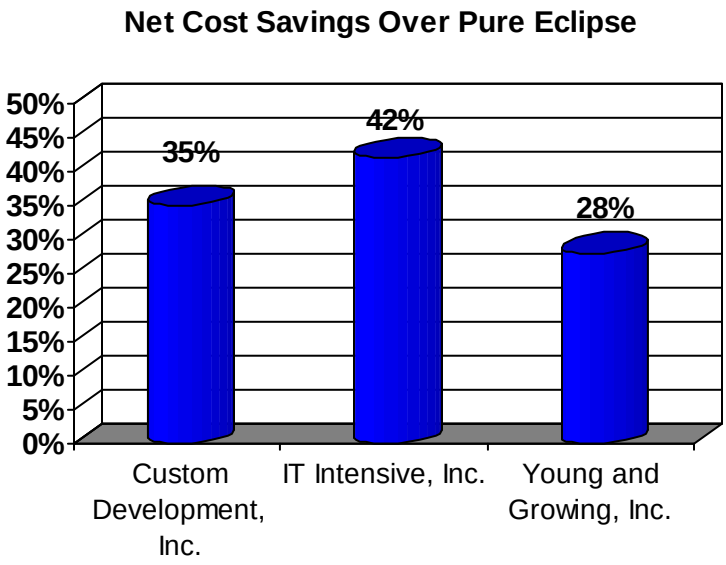
The study measured development cost, time to completion, and resulting application quality. In all situations, all three commercial IDEs (Genuitec's MyEclipse, Embarcadero JBuilder, and IBM® Rational® Application Developer) were found to offer substantial development cost savings and project quality improvements over the baseline free Eclipse distribution.



Cost Comparison of JBuilder versus Eclipse for Existing Projects

For typical software development organizations, these percentages translate into substantial net hard dollar savings in terms of software development personnel, time and quality. For the representative organizations used in this study, the return on investment (ROI) of acquiring JBuilder ranged from 90:1 to 165:1. That is, for every dollar spent on JBuilder, an organization can expect a return of $90-$165 in savings through developer productivity and improved quality.

JBuilder's approach is weighted toward increased developer and team productivity as well as application quality in heterogeneous development and deployment environments (e.g.: supporting multiple technologies for SCM, bug tracking, project management, web server, and application server).



SUMMARY

Using offshore development resources is no longer a matter of "if", but "when" for most organizations. Cost, competition, availability of skilled resources, and time to market are a few of the drivers behind this move. But moving development offshore isn't without its challenges and issues. Understanding these challenges and how to address them determines how successful the offshore team will be, which in turn affects how successful the whole organization will be. One key to this success is ensuring that the development teams, whether on or off shore, have the tools they need to achieve a superior level of productivity and deliver the highest quality code possible to meet the needs of the business. JBuilder's rich set of features and tools meet this need.

APPENDIX

MEETING REMOTE DEVELOPMENT NEEDS WITH JBUILDER

Pain/Problem	Goal	JBuilder 2008 Features
Offshore teams frequently inherit legacy code bases which they are not familiar with.	Fast, meaningful comprehension of existing projects/code.	LiveSource, Audits/Metrics, Code Archeology.
Offshore teams suffer from high turnover.	Create consistency and highly maintainable code/projects. Enable easy transfer of project/code knowledge.	Application Factories, Tagging, Scripts.
Offshore teams often are responsible for resolving hard to find performance issues with unfamiliar code.	Easily and quickly identify and correct performance bottlenecks in legacy or unfamiliar code.	Performance tools in JBuilder, Request Analyzer. Thread Debugger.
Difficult for domestic distributed and offshore teams to work and collaborate together seamlessly. Results in slow and failed projects and decreased quality.	Create fully collaborative team environment within and across teams and geo locations.	TeamInsight.
Hard to monitor and progress of remote/Offshore teams.	Get up to the minute status and reports on remote team performance/projects.	TeamInsight Portal
Need to know what code is covered when the application runs, even if you aren't familiar with the code.	Find uncovered code when writing unit tests. Understanding exactly which classes, methods, and lines of code have not been used allows you to modify your test plan to cover all areas of the code.	Code Coverage.

Pain/Problem	Goal	JBuilder 2008 Features
When code is developed over time by several developers, portions of code and sometimes whole methods are no longer used, but remain dormant in the code. This type of dead code makes an application larger and more difficult to understand.	Locate dead code easily so that you can clean up and simplify your application. Clarity of code makes it easier to read and more reusable.	Code Coverage.
Developers must be able to analyze code running remotely.	Enable a remote team who is responsible for testing or maintaining an application to collect performance data on an application running on machine they don't have physical access to.	Remote agent support in performance analysis tools.
Need to analyze all levels in a Web application, even if functionality for each level comes from different developers or teams.	CPU performance analysis of J2EE protocols. Want to obtain precise drill-down information about performance bottlenecks in any one of JDBC, JNDI, CCI, RMI, EJB, JSP, JMS, or WSVC protocols. This tool also provides protocol-specific quality analysis of unclosed resources, exceptions, and other potential issues.	Request Analyzer feature in the performance analysis tools.
Need an effective, efficient way for a person or team to audit code that is not familiar with the entire code base.	Want to insure general code quality and conformance to coding standards when it is written by various development teams around the world.	Audits capability in UML modeling. Over 200 (209) specific audit values reported. Code audits feature provides a wide variety of audits, ranging from design issues to naming conventions, along with descriptions of what each audit looks for and how to fix violations.

Pain/Problem	Goal	JBuilder 2008 Features
Difficult to determine the complexity of a code base that was developed by one team, but may be maintained by another. Also, with multiple teams working on a large project, it's difficult to get an overall measure of how complex the code base is when it comes together.	Want to get a handle on the complexity of the code base which is developed by multiple teams, and in some cases includes open source code. Want metrics to help evaluate object model complexity and quantify code. Want clear indications as to which parts of the code are candidates for redesign in order to reduce complexity.	A wide variety of metrics, ranging from lines of code to comment ratio are available within the UML modeling feature set. Determine which code needs to be redesigned, or use the results to create reports and compare the overall impact of changes in a project. Nearly 100 (90) specific metric values reported.
Not easy for developers to exchange information about project status, code updates, etc. while working within the IDE on a particular piece of code or issue.	Need an easy, efficient way for developers to communicate and share project information. Better communication between developers.	Peer-to-peer collaboration features allow two or more users to collaborate across a local area network (LAN) and send data. Peers are discovered automatically when they are on the same LAN.
Need to be able to communicate easily with an entire team of developers, or a specific sub-team about project status, issues, etc. whether they are local or remote.	Want to be able to create and manage groups/lists of team members within the development environment to enable fast, easy communication without having to specifically invite each individual to a collaboration room	Use contact groups to organize your peer list. For example, you could create a group of people working on specific product features. Then, instead of selecting each member individually, you can select the group to open a session. You can add and remove groups and peers within groups. One peer can appear in multiple groups.

Pain/Problem	Goal	JBuilder 2008 Features
Need to be able to easily share projects with an entire team of developers, or a specific sub-team whether they are local or remote.	Quickly and easily share complete project workspaces with team members without having to provide a lot of information about how to locate or import the project.	Share projects through a repository. To share a project, you can send the VCS link to a peer. The VCS link contains an identifier for the VCS plug-in, a reference to the VCS location for the project, and the name of the project. Your peer opens the VCS link to automatically check out the project locally.
Developers must be able to test code running remotely.	Enable a remote team who is responsible for testing or maintaining an application to run test scenarioson an application running on machine they don't have physical access to.	Ability to run JUnit tests remotely.
Hard to monitor the progress of remote/offshore teams.	Get up to the minute status and reports on remote team performance/projects.	Monitor activity in source code repository for project, track recent check-ins via TeamInsight Project Portal.
Hard to monitor the progress of remote/offshore teams.	Want to insure that quality metrics.	Monitor quality metrics including tables of bugs by severity, by product area, by owner, newest bugs, and bug find/fix rates via TeamInsight Project Portal.
Hard to monitor the progress of remote/offshore teams.	Want to insure project is tracking to schedule, and where potential delays or schedule risks are before they happen.	Monitor team velocity via live burn-down charts
Hard to monitor the progress of remote/offshore teams.	Want to insure project is tracking to schedule, and where potential delays or schedule risks are before they happen.	Monitor team progress against committed features, feature-by-feature in the TeamInsight portal.

Pain/Problem	Goal	JBuilder 2008 Features
Hard to monitor the progress of remote/offshore teams.	Want to insure project is tracking to schedule, and where potential delays or schedule risks are before they happen.	Monitor continuous integration builds, track failed builds to identify root-cause in the TeamInsight portal
Hard to monitor the progress of remote/offshore teams.	Want to track status and progress of individual developers possibly for management purposes, but also for early detection of possible schedule or quality issues before they happen.	Single-pane view of individual's project responsibilities: Assigned Tasks, Requirements Owned, Requirements Tracked, Assigned Bugs, Reported Bugs, and Code To-Dos

RESOURCES AND REFERENCES

Using an Agile Software Process with Offshore Development by Martin Fowler
http://martinfowler.com/articles/agileOffshore.html#TheFutureOfOffshoreAndAgile

25 Rock Solid Tips to Supervise Offshore Development
http://blog.softwareprojects.org/tips-supervise-offshore-development-447.html

silicon.com: Offshore cost-advantage to last for 20 years
http://www.silicon.com/silicon/services/offshoring/0,3800004877,39166425-1,00.htm

Inc.com: Why Some Companies Are Insourcing Software Development?
http://www.inc.com/inc5000/2008/articles/insourcing-software.html

Open Source Software: Your Company's Legal Risks
http://www.technewsworld.com/story/64378.html?wlc=1222273356

Open Source Software: Potential Costs Savings, Likely Culture Shift
http://businessinnovation.cmp.com/bizagility/feat_bizagility_07082008.jhtml

TechRepublic: 10 ways to increase the productivity of your programmers
http://downloads.techrepublic.com.com/abstract.aspx?docid=378321

Wikipedia: List of Web Application Frameworks
http://en.wikipedia.org/wiki/List_of_web_application_frameworks

Embarcadero Technologies, Inc. is a leading provider of award-winning tools for application developers and database professionals so they can design systems right, build them faster and run them better, regardless of their platform or programming language. Ninety of the Fortune 100 and an active community of more than three million users worldwide rely on Embarcadero products to increase productivity, reduce costs, simplify change management and compliance and accelerate innovation. The company's flagship tools include: Embarcadero® Change Manager™, CodeGear™ RAD Studio, DBArtisan®, Delphi®, ER/Studio®, JBuilder® and Rapid SQL®. Founded in 1993, Embarcadero is headquartered in San Francisco, with offices located around the world. Embarcadero is online at www.embarcadero.com.