APPLICATION PERFORMANCE MANAGEMENT WITH

JENNIFER®

GRATEX INTERNATIONAL TAKES JENNIFER FOR SOFTWARE DEVOLOPEMENT

CASE STUDY by

GRATEX INTERNATIONAL

DESCRIPTION

This document describes Gratex's Application Development and Test Team's experience with using Jennifer for diagnosing and troubleshooting application errors and performance problems. Gratex is an application developer, system integrator, and IT service provider based out of Slovakia.

Typical usage of APM tools traditionally starts in performance testing phase of application's life-cycle, in a stable and controlled environment with predefined KPIs and reasonable change management. After application has passed the testing phase, APM is continued to be used in production monitoring to troubleshoot or prevent emerging performance problems in a live system environment.

In our case, we did not have a typical environment; we did not have the luxury of a well-defined testing environment with pre-defined KPI or a controlled testing scenario. Like many busy development houses, our application development and system integration branch had several deployments each day. We constantly worked and struggled with unpredictable development problems ranging from local UI development, random WS and UI unit-tests ending with manual QA testing, all happening without any special isolation or coordination. Many components have been identified "buggy" from a performance point of view even before the author could test and confirm the functionality in the integration build. Both development and QA teams tried their best to resolve problems as fast as possible, but could not keep up with the demand. With constant discovery of application performance problems occurring and no real means to resolve them at once, we were in serious need of help. Luckily, when we were introduced to Jennifer, we found exactly what we had been looking for, a way to quickly hunt down the root-cause of a wide variety of application performance problems quickly and painlessly and solve for a proper fix all in record time.

While Jennifer helped us to solve many instances of performance problems, this document describes only a few selected issues, which we feel best describe Jennifer's strength and features.

FILE ACCESS

Customer: Bank A

System Environment, J2EE application running on WebLogic/AIX

After a newly developed application was deployed in a pre-production environment, we recommended that the customer should try Jennifer and installed a trial version of Jennifer on their server.

After using Jennifer for a short time, our customer's QA team reported to us that they immediately noticed very frequent file access by the application in question.

No.	Last Modified Time	File Name	Size (byte)	Mode	Concurrent Access Count
[0000]	2012-06-11/17:53:11	.properties	749	READ	1
[0001]	2012-08-12/19:04:34	rewrite.log	0	WRITE	1
[0002]	2012-08-22/19:08:18	_messages_ko_KR.properties	78,479	READ	393
[0003]	2012-08-22/19:08:18	_messages_ko_KR.properties	272	READ	398
[0004]	2012-08-22/19:08:20	_messages_ko_KR.properties	131	READ	4
[0005]	2012-08-22/19:09:48	_messages_ko_KR.properties	796	READ	398
[0006]	2012-08-23/00:10:44	/EmbeddedLDAPAccess.log	0	WRITE	1
[0007]	2012-08-23/00:10:44	/EmbeddedLDAP.log	0	WRITE	1
[0008]	2012-08-23/14:56:11	P-SFA01-201.lok	0	WRITE	1
[0009]	2012-08-23/14:56:15	/EmbeddedLDAP.lok	0	WRITE	1
[0010]	2012-08-23/17:29:49	_exception.log	46,624	WRITE	1
[0011]	2012-08-23/18:05:21	_authentication.log	37,847	WRITE	2
[0012]	2012-08-23/18:06:37	_security.log	119,676	WRITE	1
[0013]	2012-08-23/18:07:11	201.log	2,401,390	WRITE	1
[0014]	2012-08-23/18:13:33	access.log	4,944,463	WRITE	1
[0015]	2012-08-23/18:13:40	_daoexec.log	15,786,129	WRITE	1
[0016]	2012-08-23/18:13:40	_svcexec.log	2,441,432	WRITE	1
[0017]	2012-03-27/17:33:01	/mapping.cfg	0	READ	1

Above screenshot shows **Problem Determination, File/Socket** from Jennifer's monitoring dashboard.

Our test team simulated the same problem in our Windows environment and notified the development team. After further analysis it was determined that frequent file access had been caused by localization components (Spring Framework + our extensions), which triggered file access several times with each request, for both server side localization (JSPX) and client side localization (JSON service over the same properties files).

This finding lead to a quick fix: caching of the data of the properties files in memory during application lifetime.

Inspired by this story, our QA team tested file access also for other parts of the application (business logic and service layer), and Jennifer revealed another issue, this time in the Data Access layer.

NO	Execute Time	Gap Time	ResponseTime	Mode	File
872	16:39:13 196	213	0	ROPEN	selectTotMDiagList.vm
903	16:39:13 283	6	0	ROPEN	selectScrtMulList.vm
910	16:39:13 343	18	0	ROPEN	selectScrtMulList.vm
914	16:39:13 365	5	0	ROPEN	selectRiskMulList.vm
918	16:39:13 591	212	0	ROPEN	selectTotMDiagList.vm
971	16:39:16 267	2 472	0	WOPEN	/31

One request caused reads to several files (metadata and query definitions for data access components). Again, turning on caching for this functionality has solved the problem and lead to performance improvements.

JENNIFER: PROBLEM DETERMINATION FEATURE

There are many useful features in the **Problem Determination** tab of Jennifer, including monitoring and analysis tools for **Collection Object, File & Socket Access, HTTP Session Object, Class loading and Usage.**

JDBC MONITORING

DB access is probably more common in J2EE application than file access. Whether using high level JPA or low level JDBC, Jennifer can help identify performance issues ranging from silly query errors to complicated transaction-related bugs.

We want to share one of the silly errors we have experienced in our own data access layer, to show that even when developers are using sophisticated API components, mistakes can happen though. Our case involves the usage of the quite safe Spring JDBC API, and shows that even that can be used “incorrectly”.

Looking at Statistics/Application/Daily Exceptions, you can spot easily several anomalies here:

Occurrence Counts	Occupancy Ratio	Exception
44,352	86.231	JDBC ResultSet NOT CLOSED
6,656	12.941	JDBC PreparedStatement NOT CLOSED
339	0.659	UNCAUGHT APPLICATION EXCEPTION
31	0.060	APPLICATION BAD RESPONSE TIME
22	0.043	JDBC PreparedStatement SQL EXCEPTION
17	0.033	JDBC CallableStatement SQL EXCEPTION
17	0.033	DB Connection NOT CLOSED

After a quick code review, data access component unit test and fix, we repeated the test and again and got a much better result.

Occurrence Counts	Occupancy Ratio	Exception
15	7.009	JDBC ResultSet NOT CLOSED

For JDBC monitoring, test teams usually rely on third-party tools with custom driver tracing and DB level monitoring. In these cases, the use of different techniques and correlation of several logs are needed to get the same results as shown in Jennifer. Jennifer simplifies many DB access related problem detections and analysis under one umbrella.

JENNIFER: DATA ACCESS MONITORING

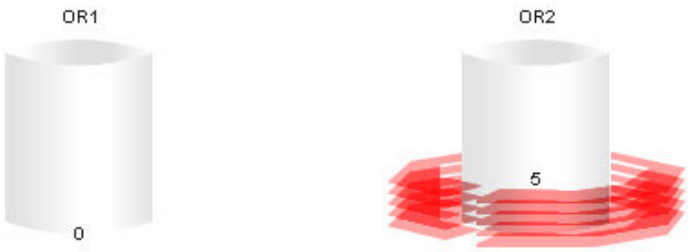
JDBC monitoring involves the collection of an excessive amount of data therefore Jennifer agent uses SQL hash values to minimize the amount.

Another useful feature is the **CRUD Matrix** screen. It shows the relation between the application services and the database tables and functions that have been created, read, deleted and updated.

LOCKED THREAD ISSUE

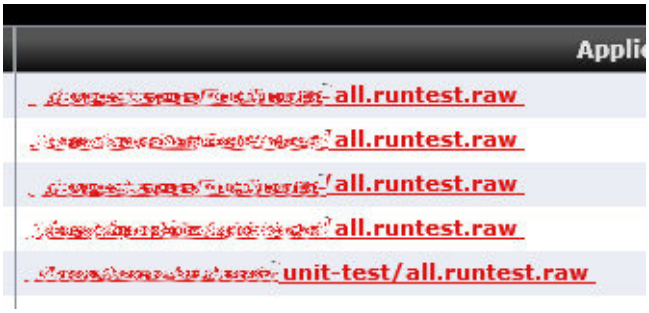
Very common and quite deadly when manifested in production runtime, thread lock issues are application developer's worst nightmare. In the worst case, containers cannot reuse the thread and with more buggy requests, applications may experience DOS.

In Jennifer's dashboard, the **Real-time Throughput Speed Meter** constantly monitors for potential application thread problems.



Out of two application servers shown above, OR2 is clearly having performance issues with response time for service requests prolonging indefinitely.

With just a few clicks more details about the troubled service requests can be seen.



As well as a detailed stack trace …

Thread Name	[ACTIVE] ExecuteThread: '4' for queue: 'weblogic.kernel.Default (self-tuning)' / 0x75d27700 (isAlive: true, isDeamon: true, isInterrupted: false, isSuspended: false)
Application	/[redacted]l.runtest.raw
HTTP Method	GET
Call Time	2012-12-11/11:54:41
Elapsed Time	4,044.61 seconds
Status	APPRUN (After the initalizing task, now executing an application logic)
	org.springframework.web.servlet.FrameworkServlet.processRequest(FrameworkServlet.java:681) org.springframework.web.servlet.FrameworkServlet.doGet(FrameworkServlet.java:574) javax.servlet.http.HttpServlet.service(HttpServlet.java:707) javax.servlet.http.HttpServlet.service(HttpServlet.java:820) weblogic.servlet.internal.StubSecurityHelper$ServletServiceAction.run(StubSecurityHelper.java:227) weblogic.servlet.internal.StubSecurityHelper.invokeServlet(StubSecurityHelper.java:125) weblogic.servlet.internal.ServletStubImpl.execute(ServletStubImpl.java:300)

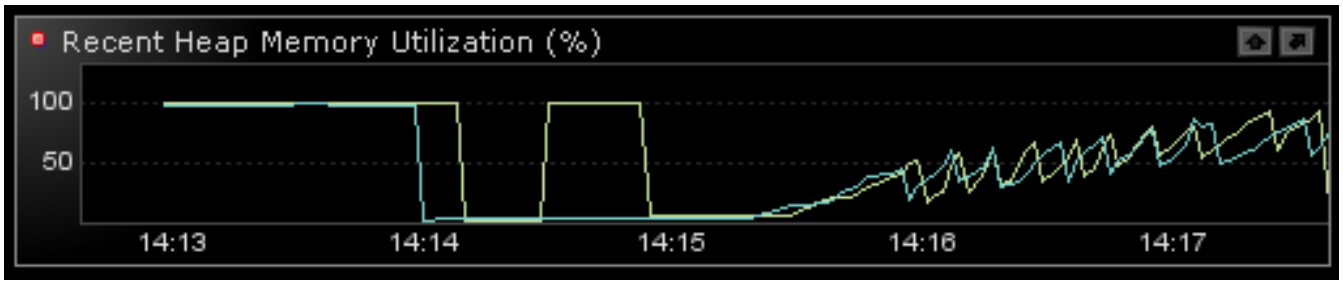
And crosschecked in WebLogic Admin Console ...

[STUCK] ExecuteThread: '0' for queue: 'weblogic.kernel.Default (self-tuning)'	73289	weblogic.servlet.internal.ServletRequestImpl@41fb1730[GET [redacted]unit-test/all.runtest.raw HTTP/1.1 User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:10.0.2) Gecko/20100101 Firefox/10.0.2 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8 Accept-Language: en-us,en;q=0.5 Accept-Encoding: gzip, deflate ECID-Context: 1.004o92hNDvi4ulYFLrJN8A0000bW000v9a;kXjE1ZDLIPJNjERPnV9OkGVBpPRBpUPQoT8T_JSTj1ORgUASpUOT_JSTi2TOrG Connection: Keep-Alive Proxy-Client-IP: 192.168.136.171 X-Forwarded-For: 192.168.136.171 X-WebLogic-KeepAliveSecs: 30 X-WebLogic-Request-ClusterInfo: true x-weblogic-cluster-hash: g9YWHa/pimylWyFtkEJNhuMXFUE]

Using Jennifer we determined that the problem was located somewhere between Spring MVC, Apache Tiles2 and the WebLogic Servlet Container. After further analysis this issue was resolved by a reconfiguration of the handler responsible for pages not found.

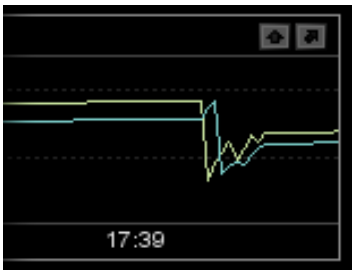
Recent Heap Memory Utilization

Another feature in Jennifer we found very useful is **Memory Utilization**.



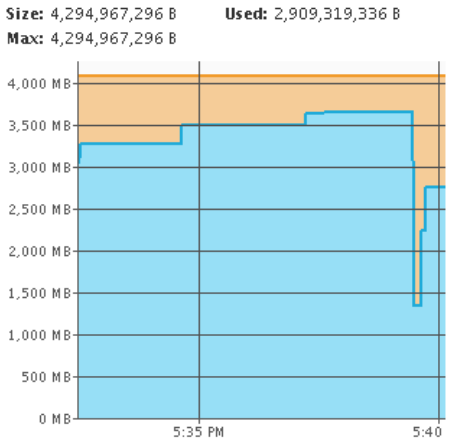
Above example shows another real-life situation from integration servers. Two lines indicate our minimal cluster (2 servers) having 100% heap memory utilization. Admin initiated restart (14:14) and 2 minutes after restart, the application used 100% of heap memory again.

After generating a heap dump (we used good old VisualVM for this, but Jennifer supports this as well), and analyzing the data (using IBM heap dump analyzer), the problem was identified in Commons Logging. Documented as known issue with redeployment, the problem was quickly fixed by explicit calls to LogFactory.release() on application shutdown.



After the fix, memory was released after redeployment (5:40), and the resolution was confirmed with Jennifer.

AND RE-VERIFIED BY VISUALVM

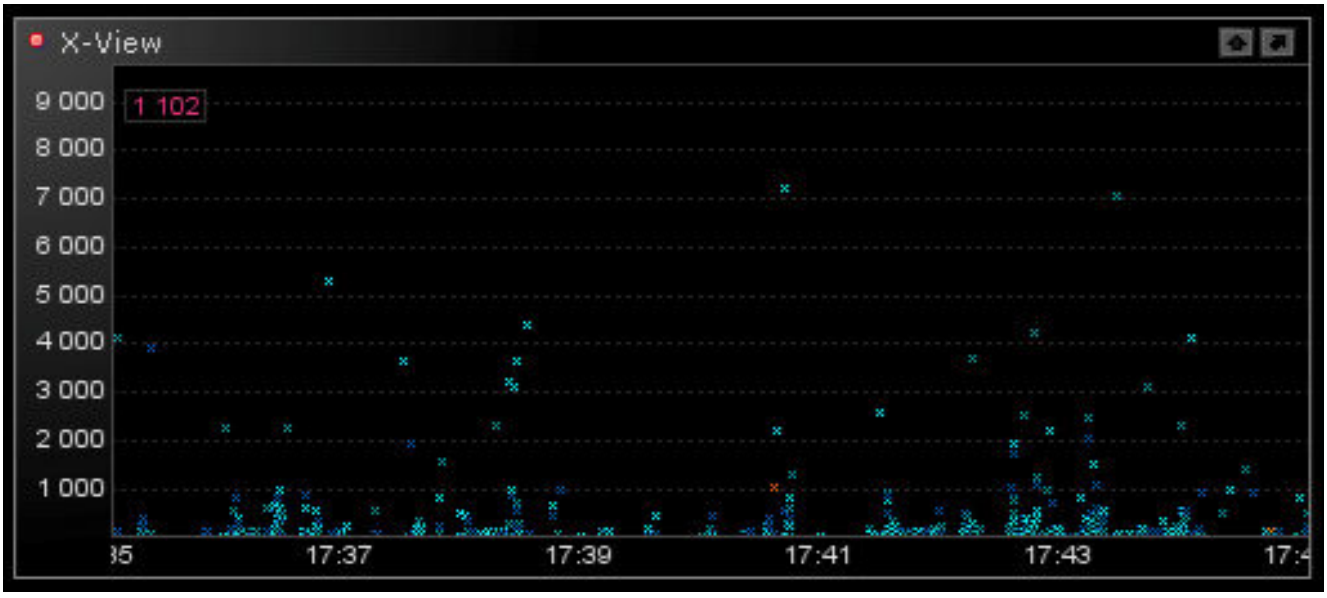


SLOW SERVICE

In our architecture most of the functionality is exposed as services (SOAP or REST/JSON) using several layers (Controllers, Services, BL components, Data Access layer, RDBMS etc.) and several third-party and open source components or containers (JEE Container, Jackson, Spring, JDBC drivers etc.). Usually a service performance problem is spotted by performance tests or front end users and labeled as “slow service”. Finding the root cause in the jumble of multiple layer’s components from various vendors and open sources can be quite complicated and often requires a lot of time and effort. This is the kind of situation where Jennifer shines brightest.

Firstly, long response times can be easily spotted in **X-View.** From here, we drilled down to individual service transactions and monitored the response times for each process.

One of our developers reported a response time greater than 5 seconds from his local HTTP proxy (Fiddler) measurements.



Using **X-View** these requests are easily measured for response time and drilled down to details in the Profiling Tab.

```
----------------------------------------------------------------------------------------------
[ NO ][ START_TIME ][ GAP][CPU_T]
----------------------------------------------------------------------------------------------
[0015][12:28:29 861][ 185][   0] SQL-PREPARE_STMT{4}
                    SELECT T.*
                    FROM [illegible]_TP_ALL T
                    param1:[]

[0016][12:28:29 862][   1][   0] SQL-EXECUTE-QUERY{4} [1 ms]
                    param1:[] param2:[]
[0017][12:28:29 862][   0][   0] FETCH[2/278]
[0018][12:28:35 725][5 863][   0] COMMIT
```

This Jennifer's **X-Viewer** shows times and gaps between individual operations. Going deeper into the internal log files (matching the same select and transaction ID), we identified the mysterious gap:

```
[2012-12-10 12:28:29] [[ACTIVE] ExecuteThread: '17' for queue: 'weblogic.kernel.Default (self-tuning)'] DEBUG [JdbcTemplate]: Executing prepared SQL statement
[2012-12-10 12:28:29] [[ACTIVE] ExecuteThread: '17' for queue: 'weblogic.kernel.Default (self-tuning)'] DEBUG [QueryExecutor]: Traced SQL query:
SELECT T.*
FROM [illegible]_TP_ALL T
[2012-12-10 12:28:29] [[ACTIVE] ExecuteThread: '17' for queue: 'weblogic.kernel.Default (self-tuning)'] INFO  [DataMapper]: Start of mapping of Result set to objects at: 2012-12-10, 12:28:29
[2012-12-10 12:28:29] [[ACTIVE] ExecuteThread: '17' for queue: 'weblogic.kernel.Default (self-tuning)'] INFO  [DataMapper]: Stop of mapping of Result set to objects at: 2012-12-10, 12:28:29
[2012-12-10 12:28:29] [[ACTIVE] ExecuteThread: '17' for queue: 'weblogic.kernel.Default (self-tuning)'] INFO  [DataMapper]: Total execution time of mapping: 0 ms
[2012-12-10 12:28:29] [[ACTIVE] ExecuteThread: '17' for queue: 'weblogic.kernel.Default (self-tuning)'] INFO  [DBOperationExecutionTimeAspect]: DB operation success SELECT exec. time at: 1 ms
[2012-12-10 12:28:29] [[ACTIVE] ExecuteThread: '17' for queue: 'weblogic.kernel.Default (self-tuning)'] INFO  [EhcacheMap]:
**************************************************************
FULL LOAD STRATEGY : full load of [illegible]TypeVo entity, records count 2, rec. versions count 2
**************************************************************
[2012-12-10 12:28:29] [[ACTIVE] ExecuteThread: '17' for queue: 'weblogic.kernel.Default (self-tuning)'] DEBUG [EhcacheMap]: FULL LOAD STRATEGY : Cache hit class [illegible]TypeVo (search by business ids [2])
[2012-12-10 12:28:29] [[ACTIVE] ExecuteThread: '17' for queue: 'weblogic.kernel.Default (self-tuning)'] DEBUG [EhcacheMap]: FULL LOAD STRATEGY : Cache hit class [illegible]TypeVo (search by business ids [2])
[2012-12-10 12:28:29] [[ACTIVE] ExecuteThread: '17' for queue: 'weblogic.kernel.Default (self-tuning)'] DEBUG [EhcacheMap]: FULL LOAD STRATEGY : Cache hit class [illegible]TypeVo (search by business ids [2])
[2012-12-10 12:28:29] [[ACTIVE] ExecuteThread: '17' for queue: 'weblogic.kernel.Default (self-tuning)'] DEBUG [EhcacheMap]: FULL LOAD STRATEGY : Cache hit class [illegible]TypeVo (search by business ids [2])
[2012-12-10 12:28:29] [[ACTIVE] ExecuteThread: '17' for queue: 'weblogic.kernel.Default (self-tuning)'] DEBUG [EhcacheMap]: FULL LOAD STRATEGY : Cache hit class [illegible]TypeVo (search by business ids [2])
[2012-12-10 12:28:29] [[ACTIVE] ExecuteThread: '17' for queue: 'weblogic.kernel.Default (self-tuning)'] DEBUG [EhcacheMap]: FULL LOAD STRATEGY : Cache hit class [illegible]TypeVo (search by business ids [2])
...
~490x the same

[2012-12-10 12:28:34] [[ACTIVE] ExecuteThread: '17' for queue: 'weblogic.kernel.Default (self-tuning)'] DEBUG [EhcacheMap]: FULL LOAD STRATEGY : Cache hit class [illegible]TypeVo (search by business ids [2])
```

This showed some problems in the Data Access Layer and EH-Cache usage for "code tables lookup", which had been repeated 500 times. The development team was alerted of this finding, including details from Jennifer and logs, and they developed a fix for this issue right away. After applying the fix the final duration of the service was less than 1 second.

Low Impact

When choosing a performance monitoring tool, one of the important things is low overhead and impact on the tested application and infrastructure. We can say that Jennifer succeeded to run, work and provide results in situations where other tools had failed to monitor or had burdened the target system with intolerable overhead for a reasonable monitoring. In fact, we have not experienced any significant overhead from Jennifer's agent to the target system and left it "turned-on" on our application servers since it was installed.

Resume

Our development and QA teams have many years of experience with troubleshooting, inspecting and bug-fixing, and we have used (and are still using) many software solutions to help us with our day-to-day tasks.

Besides these tools, Jennifer has proven to be superior to the others and solidified its important role as useful, intuitive and professionally crafted software, combining many useful features in one place.

Considering Jennifer for both production and end user testing is definitely a good choice, but we can recommend it also as a valuable tool during the development phase.

We are looking forward to continuing to experiment with Jennifer, next time in QA stress and load test environment.

To get more information about Gratex's experience with Jennifer please contact

Gratex International, Software Development Division

About Gratex International

Gratex International belongs to the largest and most successful IT companies on the Slovak market, where it has been active since 1991. We employ almost 300 highly skilled employees, 70 % of whom work in software development area. More than 30 % of our employees are certified specialists. Over the years we have recorded a continual growth, ranking Gratex International among the top IT companies in Slovakia according to the Trend magazine - the leading economic weekly.

Our solutions are based on the most progressive software technologies of world IT leaders. We are registered IBM Premier Business Partner, ORACLE Gold Partner, MICROSOFT Gold Certified Partner, HP Gold Preferred Partner, SYMANTEC, TCOM, SAP, CISCO, Samsung SDS. With some of these companies we are registered as their first partner in Slovakia, and we enjoy the highest certification level possible to achieve.

We also work with the Open Source Frameworks including Spring Framework, Dojo Toolkit, PostgreSQL, MongoDB and many others.

www.gratex.com

About Jennifersoft

JenniferSoft, Inc. is a software development company with expertise in system performance monitoring and problem resolution. With experience in enterprise system planning and consultation, JenniferSoft provides Application Performance Management (APM) solutions and services for companies' enterprise web-systems. JenniferSoft's APM solution, JENNIFER®, specializes in JAS (WebSphere, WebLogic, Resin, GlassFish, JBoss, Tomcat, etc.) performance monitoring and supports different types of Operating Systems including Windows, all types of UNIX (IBM AIX, Oracle Solaris, HP-UX), IBM i and especially z/OS. JenniferSoft's APM solution, JENNIFER®, provides efficient real-time system monitoring, practical performance problem diagnosis and troubleshooting, and effective performance management for all enterprise web-based systems on the market.

www.jennifersoft.com