APPDYNAMICS

An AppDynamics Business White Paper

MASTERING APPLICATION PERFORMANCE IN FINANCIAL SERVICES

The Impact of Transaction-based Application Performance Management

Managing applications in the world of Financial Services (FS) is different from any other industry. Regulations play a pivotal role in the development and support of banks' business-critical applications, and these regulations often make application management more difficult. However, because many businesses depend on online transactions for their revenue, application performance and throughput has become more important than ever.

In this paper we'll explore a new way to master the performance of FS applications by monitoring and managing their business transactions. Focusing on the performance and execution of transactions – that is, the way your users interact with your application – you can become more proactive and efficient at troubleshooting and managing application performance. As a result, you'll save millions on revenue and productivity, and you will also gain the loyalty of your customers.

In this white paper, we propose that transaction-based application performance monitoring can help financial organizations increase revenue according to this formula:

$$\begin{array}{ll} & \text{Faster transactions + Faster time to market + Reduced MTTR} \\ \text{X} & \text{More stable platform} \\ \hline & \text{Higher profit} \end{array}$$

"Businesses who boosted customer retention rates by as little as 5% saw increases in their profits ranging from 5% to a whopping 95%."

The speed of transactions and the stability of the system as a whole play an integral role in customer retention and, therefore, revenue. This means by speeding up transactions, you can increase your margins and improve your relationship with your customers. But before we get into the how of transaction-based APM, let's look at the why – why is it that transaction speed (or rather, response time) has such a significant impact on profitability?

1. Customers Hate to Wait

Losing a customer is a bad thing no matter the circumstances. Losing a customer due to poor application performance and stability is preventable and should never happen! If you lose customers, you either need to win them back or attract new customers.

Fred Reichheld of Bain & Company reports that:

- Over a five-year period businesses may lose as many as 1/2 of their customers
- Acquiring a new customer can cost 6 to 7 times more than retaining an existing customer
- Businesses who boosted customer retention rates by as little as 5% saw increases in their profits ranging from 5% to a whopping 95%

Based on this research, you should do everything in your power to retain your existing customers. Providing a great end user experience and level of service is what every customer expects.

2. Customer Loyalty = Revenue Growth

On the flipside, better performance means more customer loyalty and, as a result, revenue gains. The Net Promoter methodology, developed by Satmetrix in cooperation with Bain & Company and Fred Reichheld, is a standard way to measure customer satisfaction and loyalty. In their Net Promoter whitepaper, Satmetrix discovered a direct correlation between high customer loyalty scores and the rate of revenue growth for those companies. The paper showed that the higher the customer loyalty score a company achieved, the higher their rate of revenue growth over a 5-year period.

"It is imperative that IT organizations take a user-centric, or rather, transaction-centric approach to managing application performance."

With applications playing a dominant role as the most common interaction between company and customer, it is imperative that customers have a great experience every time they use your application. Slow transactions, errors, and unavailable platforms leave customers dissatisfied and will reduce your loyalty score. Over time, this will have a significant impact on revenue.

So if we accept the premise that performance should be top-of-mind for anyone with a critical banking or FS application, what do we do next? How do we improve our application management strategy to prevent loss of revenue and improve customer loyalty? The answer: by adopting a transaction-based approach to application performance management.

3. Transactions = Money

Transactions are the lifeblood of banking. From making an online payment or converting currency to buying or selling stock, just about everything a bank does involves transactions. Furthermore, a significant portion of banks' revenue comes from transaction fees for activities ranging from ATM withdrawals to currency conversion and credit card usage. For these fee-based transactions, the faster you can ring the cash register (response time of business transactions), the more money you will make and the better likelihood that your customer will come back to you for their next transaction.

With this in mind, it is imperative that IT organizations take a user-centric, or rather, transaction-centric approach to managing application performance.

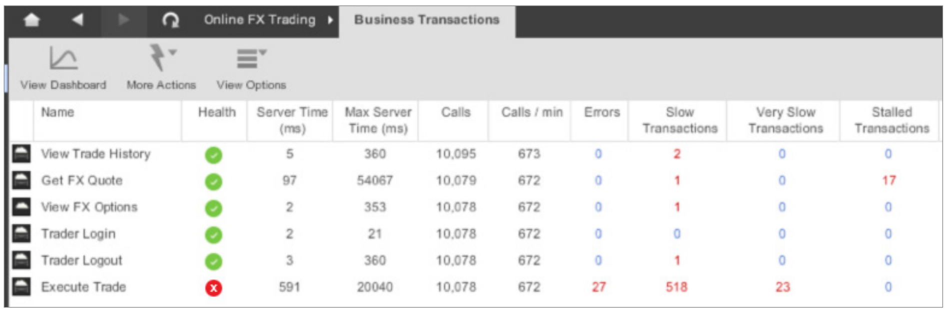
“Make sure you have deep visibility into the application code execution associated with each transaction.”

A TRANSACTION-BASED APPROACH TO APPLICATION PERFORMANCE MANAGEMENT

Now that we understand why transactions and performance are so important to revenue, let’s take a look at the best way to improve performance and increase revenue: transaction-based application performance management (APM).

1. Monitor All Real User Transactions

Monitor your applications from your customers’ perspective. The end users of your applications submit transactions to your systems for processing. Monitor all of these transactions throughout their entire lifecycle (end-to-end) so that you know exactly how long each transaction takes and where exactly they slow down or fail. Sampling transactions or relying upon synthetically generated transactions is not good enough. Monitor all real user transactions!



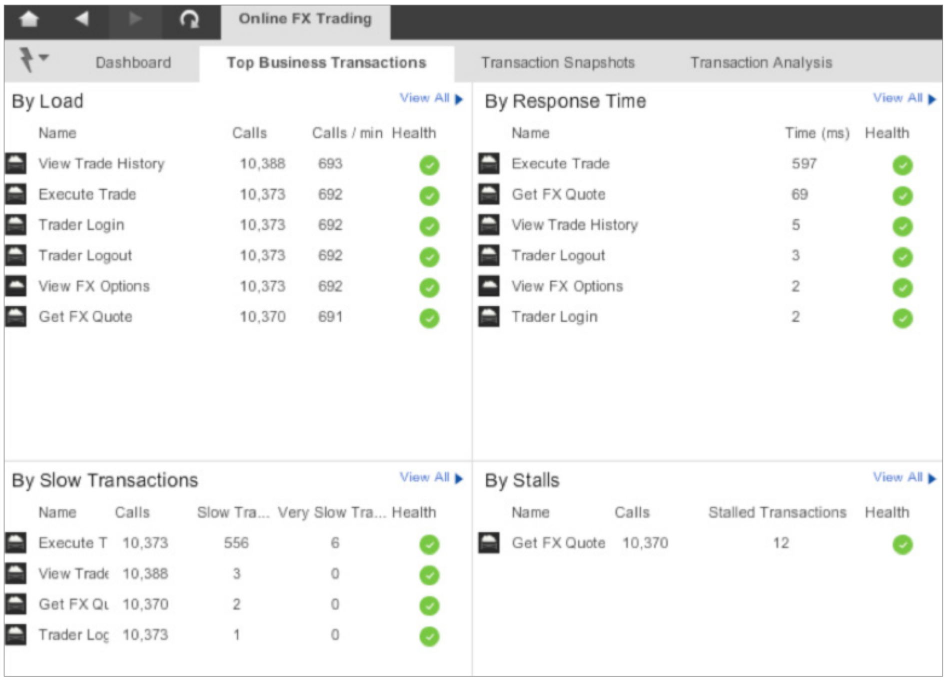
Name	Health	Server Time (ms)	Max Server Time (ms)	Calls	Calls / min	Errors	Slow Transactions	Very Slow Transactions	Stalled Transactions
View Trade History	✓	5	360	10,095	673	0	2	0	0
Get FX Quote	✓	97	54067	10,079	672	0	1	0	17
View FX Options	✓	2	353	10,078	672	0	1	0	0
Trader Login	✓	2	21	10,078	672	0	0	0	0
Trader Logout	✓	3	360	10,078	672	0	1	0	0
Execute Trade	✗	591	20040	10,078	672	27	518	23	0

View of real user transactions and their performance for a trading platform.

Knowing which transactions your customers are using, the response time of each, and exactly where in your infrastructure any are slowing down or failing is a great start—but you need to take it a step further. Make sure you have deep visibility into the application code execution associated with each transaction. This level of visibility will allow you to identify the root cause of problems in your applications rapidly.

“A major investment bank has an MTTR goal of 15 minutes for all of its trading applications.”

2. Improve Performance of Key Transactions
Now that you have all of this great information about your users’ activity and the performance of each individual transaction, you can use it to make your applications faster. Explore which transactions are requested the most and have the worst response times over the past month. These are the low-hanging fruit that will provide the most end user experience bang for your buck.



By Load

Name	Calls	Calls / min	Health
View Trade History	10,388	693	✓
Execute Trade	10,373	692	✓
Trader Login	10,373	692	✓
Trader Logout	10,373	692	✓
View FX Options	10,373	692	✓
Get FX Quote	10,370	691	✓

By Response Time

Name	Time (ms)	Health
Execute Trade	597	✓
Get FX Quote	69	✓
View Trade History	5	✓
Trader Logout	3	✓
View FX Options	2	✓
Trader Login	2	✓

By Slow Transactions

Name	Calls	Slow Tra...	Very Slow Tra...	Health
Execute T	10,373	556	6	✓
View Trade	10,388	3	0	✓
Get FX Qu	10,370	2	0	✓
Trader Log	10,373	1	0	✓

By Stalls

Name	Calls	Stalled Transactions	Health
Get FX Quote	10,370	12	✓

View of real user transactions and their performance for a trading platform.

3. Reduce Mean Time To Repair (MTTR)
Inevitably, every application will fail and send your support teams scrambling to restore service.

ITIL v2 defines MTTR as “the mean elapsed time from the occurrence of an incident to the restoration of service.” A major investment bank has an MTTR goal of 15 minutes for all of its trading applications. Let’s explore the typical sequence of events for an incident within a large enterprise organization.

1. An end user calls support to complain that the application is not working properly.
2. The support representative creates a problem ticket and records a description of the problem from the end user (5 – 10 minutes).
3. Support checks their run books to see if this problem has a known fix (5 minutes).
4. If this is a known issue with a known fix, then support performs the repair and service is restored within 15 minutes. If not, support must pass the ticket to the proper application support team (5 mins).

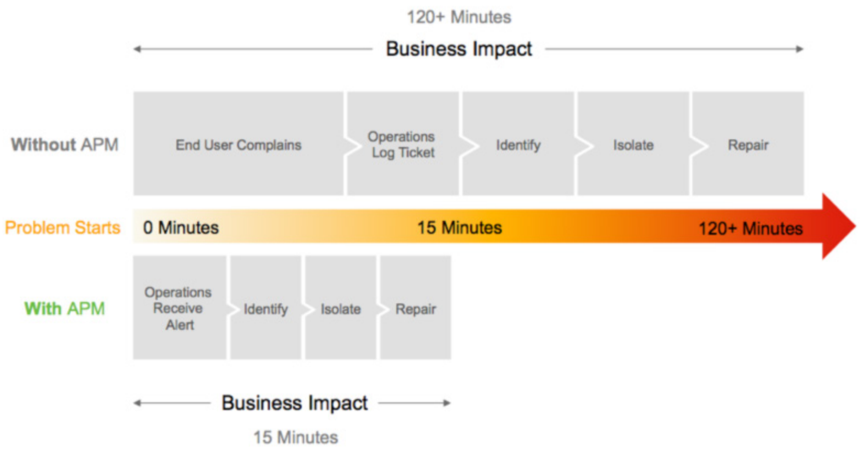
5. Application support begins their troubleshooting efforts based on the end user description of the problem as they experienced it (e.g. “When I clicked the ‘Execute Trade’ button the browser just sat there waiting and I never got a trade confirmation. Did I make the trade or not?”).
6. Application support begins manually troubleshooting their application using custom scripts, SQL queries, checking log files, etc. They determine that they need to call in the DBA and network support teams to see if anything is problematic in those silos (30-60 minutes have elapsed since the start of the incident).
7. Each support team is looking within their own silo, trying to determine if they see any issues that could impact this application. They uncover a database problem and correct the issue (45 – 90 minutes have elapsed since the start of the incident).
8. The application support team verifies that the application is working properly and confirms that service has been restored.

“Providing a great end user experience and level of service is what every customer expects.”

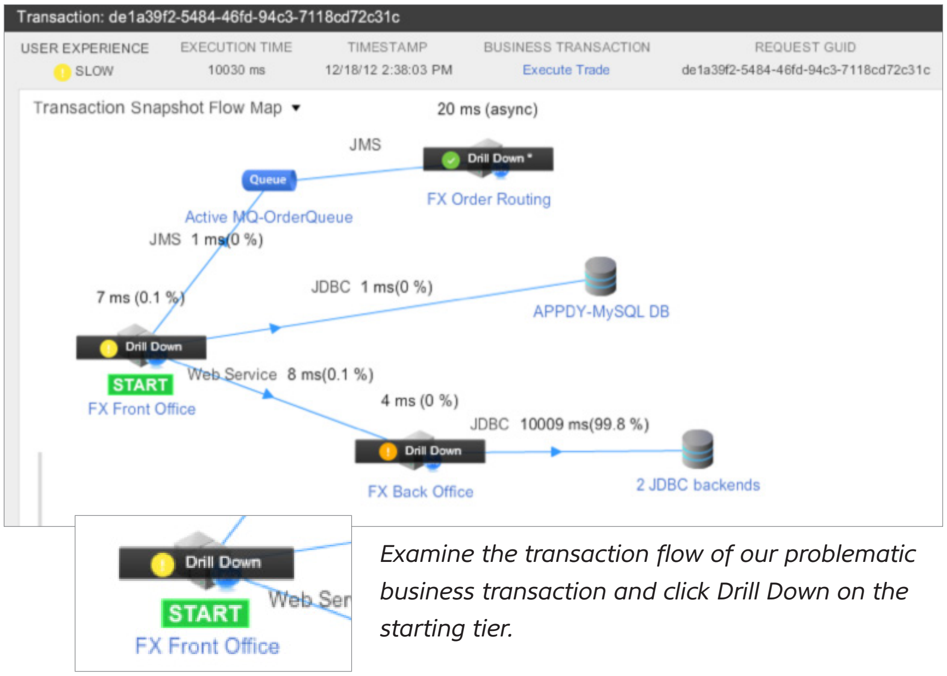
The MTTR for an event of this type typically ranges from 60–180 minutes. This is completely unacceptable to many customers, particularly any user that needs your trading platform during trading hours. Now let’s take a look at this same event when support and operations teams have access to transaction-based APM tooling.

1. Your IT Operations Center receives an automated alert indicating a stalled “Trade Execute” transaction and a problem ticket is automatically generated with this information.
2. Support checks their run books to see if this problem is something with a known fix. At the same time they click the link in the alert, which automatically directs them to the troubleshooting section of their APM tool. They see in the APM tool that the transaction stalled in the database tier (5 minutes have elapsed since start of incident).
3. End user calls support to complain that the application is not working properly and support informs them that they are already working on the problem.
4. If this is a known issue with a known fix, then support performs the repair and service is restored within 10 minutes. If not, support must pass the ticket to the proper application support team (5 minutes total elapsed since start of incident).
5. IT Operations contacts application support and DBA teams while passing along the troubleshooting information that indicates a database problem (including the exact query generated by this transaction).
6. The DBA team finds the problem and corrects the issue (15 minutes have elapsed since the start of the incident).
7. The application support team verifies that the application is working properly and confirms that service has been restored.

The same incident that took 60-180 minutes to resolve without the proper monitoring tool can be reduced to 15 minutes using transaction-based APM.

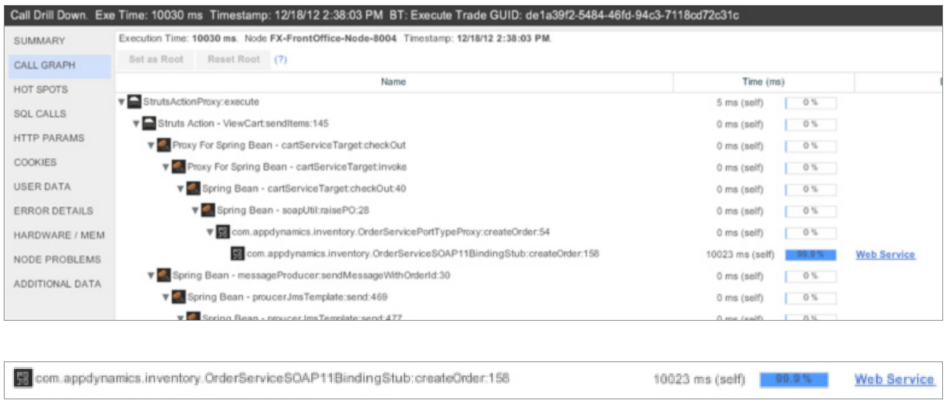


TROUBLESHOOTING WORKFLOW WITHIN AN APM TOOL (FOR THE SCENARIO DESCRIBED IN STEP 2 ABOVE)


Transaction: de1a39f2-5484-46fd-94c3-7118cd72c31c

USER EXPERIENCE	EXECUTION TIME	TIMESTAMP	BUSINESS TRANSACTION	REQUEST GUID
SLOW	10030 ms	12/18/12 2:38:03 PM	Execute Trade	de1a39f2-5484-46fd-94c3-7118cd72c31c

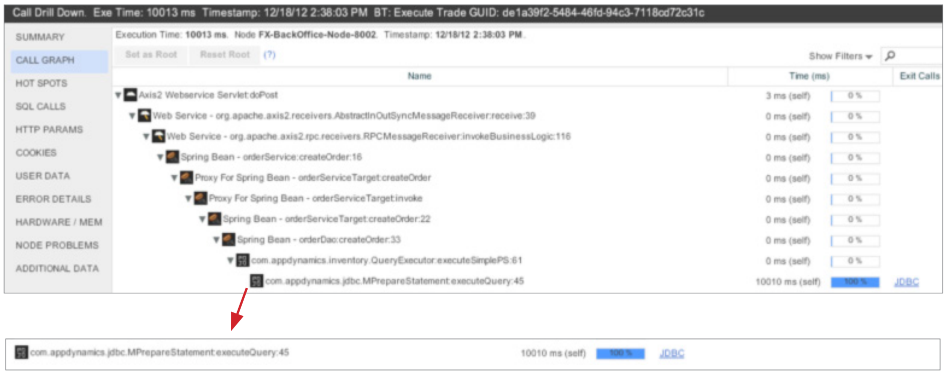
Examine the transaction flow of our problematic business transaction and click Drill Down on the starting tier.



```
Call Drill Down.  Exe Time: 10030 ms  Timestamp: 12/18/12 2:38:03 PM  BT: Execute Trade GUID: de1a39f2-5484-46fd-94c3-7118cd72c31c
SUMMARY          Execution Time: 10030 ms.  Node FX-FrontOffice-Node-8004  Timestamp: 12/18/12 2:38:03 PM.
CALL GRAPH       Set as Root   Reset Root   (?)
HOT SPOTS                                   Name                                                        Time (ms)
SQL CALLS        ▼ StrutsActionProxy:execute                                                            5 ms (self)      0 %
HTTP PARAMS        ▼ Struts Action - ViewCart:sendItems:145                                             0 ms (self)      0 %
COOKIES              ▼ Proxy For Spring Bean - cartServiceTarget:checkOut                               0 ms (self)      0 %
USER DATA              ▼ Proxy For Spring Bean - cartServiceTarget:invoke                               0 ms (self)      0 %
ERROR DETAILS            ▼ Spring Bean - cartServiceTarget:checkOut:40                                  0 ms (self)      0 %
HARDWARE / MEM             ▼ Spring Bean - soapUtil:raisePO:28                                          0 ms (self)      0 %
NODE PROBLEMS                ▼ com.appdynamics.inventory.OrderServicePortTypeProxy:createOrder:54       0 ms (self)      0 %
ADDITIONAL DATA                  com.appdynamics.inventory.OrderServiceSOAP11BindingStub:createOrder:158  10023 ms (self)  99.9 %  Web Service
                     ▼ Spring Bean - messageProducer:sendMessageWithOrderId:30                          0 ms (self)      0 %
                       ▼ Spring Bean - proucerJmsTemplate:send:469                                      0 ms (self)      0 %
                         ▼ Spring Bean - proucerJmsTemplate:send:477                                    0 ms (self)      0 %

com.appdynamics.inventory.OrderServiceSOAP11BindingStub:createOrder:158     10023 ms (self)  99.9 %  Web Service
```

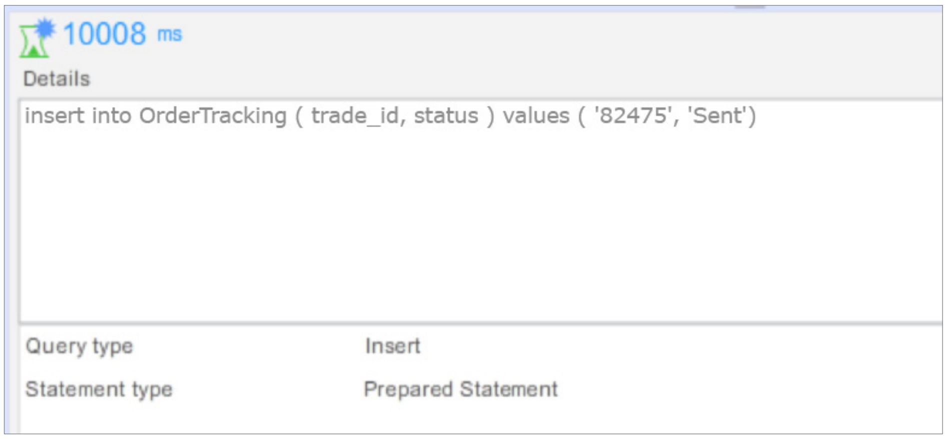
The new view shows the call graph with all custom code names and response times. The majority of time was spent in a Web Services call so we click the link to examine that call.



```
Call Drill Down.  Exe Time: 10013 ms  Timestamp: 12/18/12 2:38:03 PM  BT: Execute Trade GUID: de1a39f2-5484-46fd-94c3-7118cd72c31c
SUMMARY          Execution Time: 10013 ms.  Node FX-BackOffice-Node-8002.  Timestamp: 12/18/12 2:38:03 PM.
CALL GRAPH       Set as Root   Reset Root   (?)                                                              Show Filters ▾
HOT SPOTS                                   Name                                                        Time (ms)                Exit Calls
SQL CALLS        ▼ Axis2 Webservice Servlet:doPost                                                      3 ms (self)      0 %
HTTP PARAMS        ▼ Web Service - org.apache.axis2.receivers.AbstractInOutSyncMessageReceiver:receive:39         0 ms (self)      0 %
COOKIES              ▼ Web Service - org.apache.axis2.rpc.receivers.RPCMessageReceiver:invokeBusinessLogic:116    0 ms (self)      0 %
USER DATA              ▼ Spring Bean - orderService:createOrder:16                                      0 ms (self)      0 %
ERROR DETAILS            ▼ Proxy For Spring Bean - orderServiceTarget:createOrder                       0 ms (self)      0 %
HARDWARE / MEM             ▼ Proxy For Spring Bean - orderServiceTarget:invoke                          0 ms (self)      0 %
NODE PROBLEMS                ▼ Spring Bean - orderServiceTarget:createOrder:22                          0 ms (self)      0 %
ADDITIONAL DATA                ▼ Spring Bean - orderDao:createOrder:33                                  0 ms (self)      0 %
                                 ▼ com.appdynamics.inventory.QueryExecutor:executeSimplePS:61           0 ms (self)      0 %
                                     com.appdynamics.jdbc.MPrepareStatement:executeQuery:45             10010 ms (self)  100 %   JDBC

com.appdynamics.jdbc.MPrepareStatement:executeQuery:45     10010 ms (self)  100 %  JDBC
```

Examining the Web Services call graph we can see that the majority of response time was due to a JDBC (database) call. We can click on the JDBC link to see the offending SQL statement.



10008 ms

Details

```
insert into OrderTracking ( trade_id, status ) values ( '82475', 'Sent')
```

Query type	Insert
Statement type	Prepared Statement

The slow SQL statement is shown as the root cause of our performance problem.

4. Accelerate Time to Market
Another important revenue generation and protection concept is “Time to Market.” When a FS company launches a new product, they have a short window of time to make a high margin on that product until the competition releases their own version of the same product. New products may require new applications built from the ground up or updates to existing applications in order to sell and support them. When new code is introduced to a new or existing application, that application must go through the Software Delivery Lifecycle (SDLC). It doesn’t matter if the development techniques used are agile, waterfall, or any other development methodology; there are certain steps that will always occur. Here are the basic steps:

1. Write code
2. Test code
3. Re-write problematic code (if there is any)
4. Re-test if needed
5. Promote to production (if it breaks or is slow in production go back to step 3)

“How do you shorten a process that could take months and make sure it only takes days?”

Depending on your development methodology the preceding steps can take anywhere from days to weeks to months. The big question is: How do you shorten a process that could take months and make sure it only takes days?

The longest part of the process is figuring out if poor performance exists, if errors are introduced by the new code, and exactly what section of code is responsible. Transaction-based APM tools provide this information almost immediately after the execution of the code in any of your pre-production or production environments. The information provided by APM tools tells the developers exactly what impact their code had, and what sections of code need to be refactored to account for performance and/or stability issues.

In many FS environments, developers are prohibited from having access to production applications and servers due to the Sarbanes-Oxley Act and other regulatory requirements. This makes troubleshooting a production issue that can’t be replicated in non-production environments extremely time-consuming (or even impossible) without the help of a monitoring tool.

Conclusion
Application performance can have a significant impact on customer satisfaction and, therefore, revenue. In this white paper we’ve talked about how improved performance can increase revenue following this formula:

(Faster transactions + Faster time to market + Reduced MTTR) * More stable platform = Higher profit

$$\begin{array}{ll} & \text{Faster transactions + Faster time to market + Reduced MTTR} \\ \text{X} & \text{More stable platform} \\ \hline & \text{Higher profit} \end{array}$$

An important aspect of increasing your corporate profits – or at least minimizing loss due to misbehaving applications – is through the use of a transaction-based application performance management solution. There are many technical benefits of APM tools, but the business impact these tools have is really what is most important. If your goals as a business are to keep your customers happy, earn their repeat business, and continually improve your revenue stream and profit margins, then you need to seriously consider making an investment in transaction-based APM software.

APPENDIX A

An Example ROI Study

To see how a transaction-based approach can affect revenue, let's take a look at an ROI case study for one organization a year after deploying their APM solution. Here are the results of their exercise:

Example Use Case	Before AppDynamics	After AppDynamics	Benefits
Reduction in production downtime	99.91% availability	99.95% availability	$167,475 saved in lost revenue
Reduction in average MTTR per Severity-1 incident	5 man-days	Reduced by 45% (conservative)	$307,521 in productivity savings
Reduction in time spent identifying performance defects in pre-production	2.5 man-days	Reduced by 35% (conservative)	$320,170 in productivity savings
		Total 1st Year Savings	**$795,166**
		Projected ROI Over 2 Years	**$1,217,334**

Try it FREE at
www.appdynamics.com