Microsoft® Visual Studio®

Enabling Performance & Stress Test throughout the Application Lifecycle

March 2010

Poor application performance costs companies millions of dollars and their reputation every year. The simple challenge of releasing software that behaves predictably, consistently and responsively continues to be a big one. Internal applications with poor performance add another layer of cost - both in lost productivity and missed deadlines. Microsoft Visual Studio 2010 provides the tools to measure, improve and verify application performance under the most demanding conditions so that your application performs predictably regardless of the situation.

Poor application performance costs companies millions of dollars and their reputation every year. Why this happens is fairly straightforward – customers can't get to the website they want or they can't purchase products because the application either responds poorly or doesn't respond at all. And that's just for web-based applications. Internal company applications that suffer from poor performance waste time, and time is expensive – both in lost productivity and missed deadlines. In the past, poor performance might not have had as visible an impact but today poor performance is extremely expensive. It doesn't have to be that way though. Microsoft Visual Studio 2010 provides the tools to measure, improve and verify application performance under the most demanding conditions so that your application performs predictably regardless of the situation. Performance testing, in many cases, is not performed at all. The chief reason for this is not the time available but the cost and complexity of the tooling required to undertake performance testing.

A FAMILIAR ENVIRONMENT

One of the key benefits to using the integrated performance testing tools in Visual Studio 2010 is that they work in an environment that developers are used to. Visual Studio provides an intuitive interface for constructing Web Tests, Unit Tests and the associated Load Tests. And all tests are based on the same extensible testing framework which has matured over the previous five years into a robust and well-supported framework.

In addition to the integration with Visual Studio, customizing tests do not require you to learn a new language – they are backed by the Microsoft .NET framework. Many other industry standard performance testing tools require you to learn a new scripting language or development language and a new Integrated Development Environment (IDE). With the Visual Studio testing tools, you can be up and running in a short period of time with a minimal learning curve.

Finding Performance Problems Early

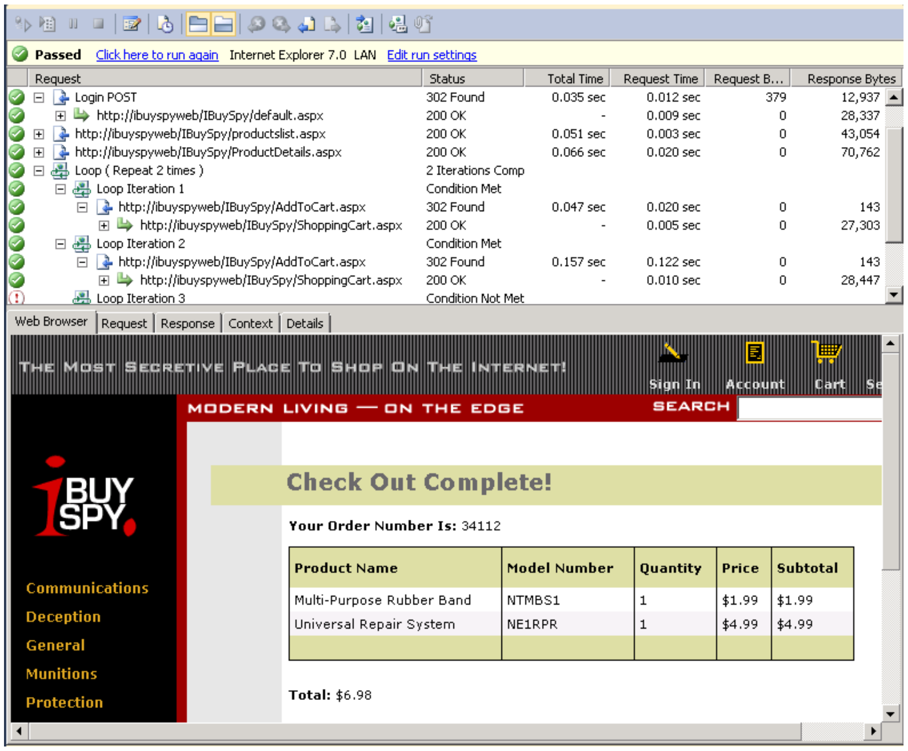
Performance problems can be introduced into applications in a variety of different ways. Often, performance issues occur because of the chosen architecture. Poor architectural structure leads to bottlenecks in code which can slow the entire application down – even the addition of faster hardware doesn't help when the software isn't architected correctly. Fixing a problem that originates with the application architecture can be difficult and costly; finding the problem early in the process can save a considerable amount of time and cost.

MODELING USER BEHAVIOR USING WEB PERFORMANCE TESTS

Web Performance tests are one of the key components of performance testing in Visual Studio 2010. This feature was introduced in Visual Studio 2005; Microsoft has taken an already solid tool and enhanced it to provide greater ease of use and better functionality.

Web Performance tests record browser traffic at the HTTP layer, which can then be run in a load test to simulate user behavior. These tests are lightweight and very efficient at generating a large amount of load.

Web Performance tests are rich in functionality. Ajax and page resources are handled automatically, and you can easily add conditions and looping to your tests without writing code. In addition, dynamic parameters, cookies and authentication are handled for you and tests can be bound to data to create flexible data driven tests. In addition to these features, Web Performance tests are extensible. If, for example, you need to add complex validation rules that are not met by the built-in rules, you can add your own. Need to do custom dynamic parameter handling for the application you are testing? You can create a plug-in using minimal code. There are numerous extensibility points which let you customize any aspect of the Web Performance testing experience. Figure 1 shows the output of a Web Performance test.


Passed Click here to run again Internet Explorer 7.0 LAN Edit run settings

Request	Status	Total Time	Request Time	Request B...	Response Bytes
Login POST	302 Found	0.035 sec	0.012 sec	379	12,937
http://ibuyspyweb/IBuySpy/default.aspx	200 OK	-	0.009 sec	0	28,337
http://ibuyspyweb/IBuySpy/productslist.aspx	200 OK	0.051 sec	0.003 sec	0	43,054
http://ibuyspyweb/IBuySpy/ProductDetails.aspx	200 OK	0.066 sec	0.020 sec	0	70,762
Loop (Repeat 2 times)	2 Iterations Comp				
Loop Iteration 1	Condition Met				
http://ibuyspyweb/IBuySpy/AddToCart.aspx	302 Found	0.047 sec	0.020 sec	0	143
http://ibuyspyweb/IBuySpy/ShoppingCart.aspx	200 OK	-	0.005 sec	0	27,303
Loop Iteration 2	Condition Met				
http://ibuyspyweb/IBuySpy/AddToCart.aspx	302 Found	0.157 sec	0.122 sec	0	143
http://ibuyspyweb/IBuySpy/ShoppingCart.aspx	200 OK	-	0.010 sec	0	28,447
Loop Iteration 3	Condition Not Met				

Check Out Complete!

Your Order Number Is: 34112

Product Name	Model Number	Quantity	Price	Subtotal
Multi-Purpose Rubber Band	NTMBS1	1	$1.99	$1.99
Universal Repair System	NE1RPR	1	$4.99	$4.99

Total: $6.98

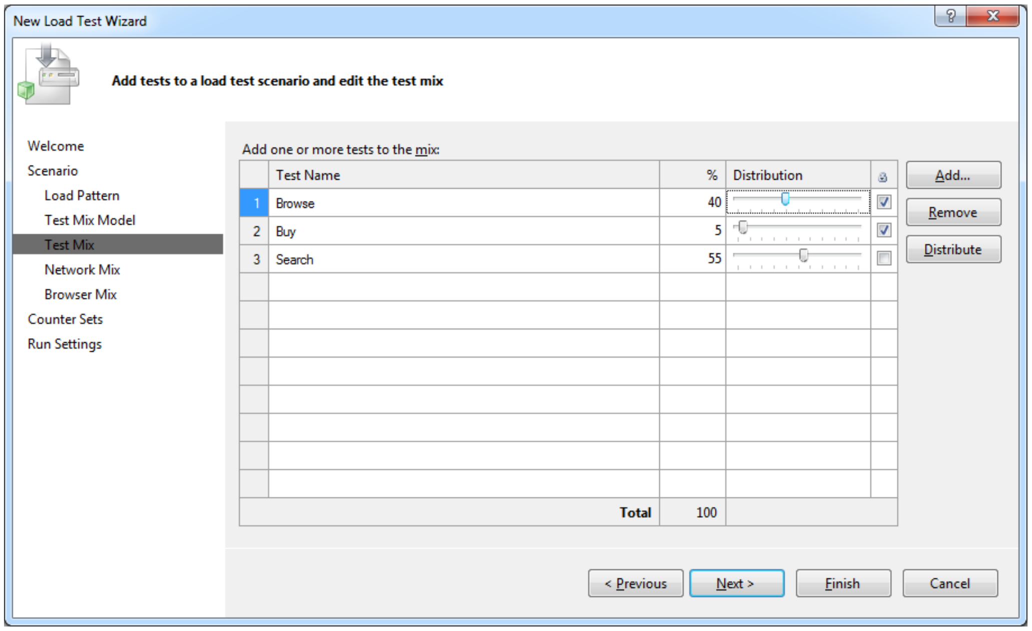
Figure 1 – Web Performance Test Output.

USING UNIT TESTS TO DRIVE LOAD

One of the great benefits of the VS 2010 performance and load testing tools is the ability to work with unit tests. This means that you don't need a full-up user interface or even all of the pieces of an application to test the performance of an application. In addition, you can perform tests earlier than normal in the process if there is a suspected performance bottleneck. This is also another benefit when working with web services: As more applications begin working with web services which back user interfaces, the performance of those services can be independently tested to ensure they perform well.

CREATING LOAD TESTS

How does performance testing work in Visual Studio 2010? First, you can create either unit tests or web performance tests or a combination of both types of tests. Next, using a simple wizard, you can construct complex load tests which have different network bandwidths, load patterns, test mixes and use a variety of different browsers. In addition, you can specify the warm up time, how long the test will run for, whether or not to use think times, and you can determine which machines you want to gather performance counters for. While the interface for selecting these options is simple, taken together they can be used to specify virtually any performance profile you want. Want some sample users on the standard 10MB/sec network bandwidth while others are using dial-up? How about gathering detailed performance information not only of the middle-tier system being tested but of the data tier, proxy server, network load balancer and various other machines along the communication path? You can set these up through the wizard as well. Figure 2 shows the Load Test Wizard.



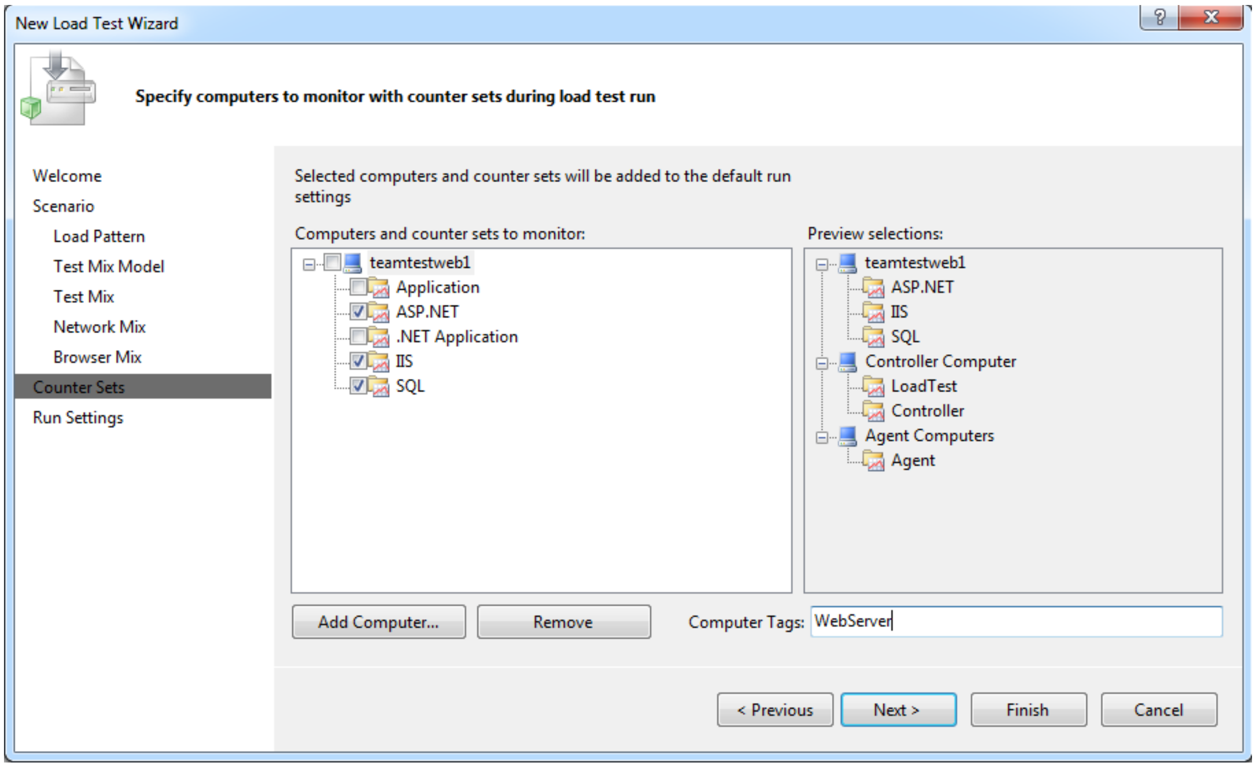
Add tests to a load test scenario and edit the test mix

Add one or more tests to the mix:

	Test Name	%	Distribution	
1	Browse	40		✓
2	Buy	5		✓
3	Search	55		
	Total	100		

Figure 2 – Set the test mix to predicted usage.

You can easily collect performance counters from the system under test using counter sets, which come with pre-configured thresholds to warn when resources are over-utilized (Figure 3).



Selected computers and counter sets will be added to the default run settings

Computers and counter sets to monitor:

- [ ] teamtestweb1
  - [ ] Application
  - [x] ASP.NET
  - [ ] .NET Application
  - [x] IIS
  - [x] SQL

Preview selections:

- teamtestweb1
  - ASP.NET
  - IIS
  - SQL
- Controller Computer
  - LoadTest
  - Controller
- Agent Computers
  - Agent

Computer Tags: WebServer

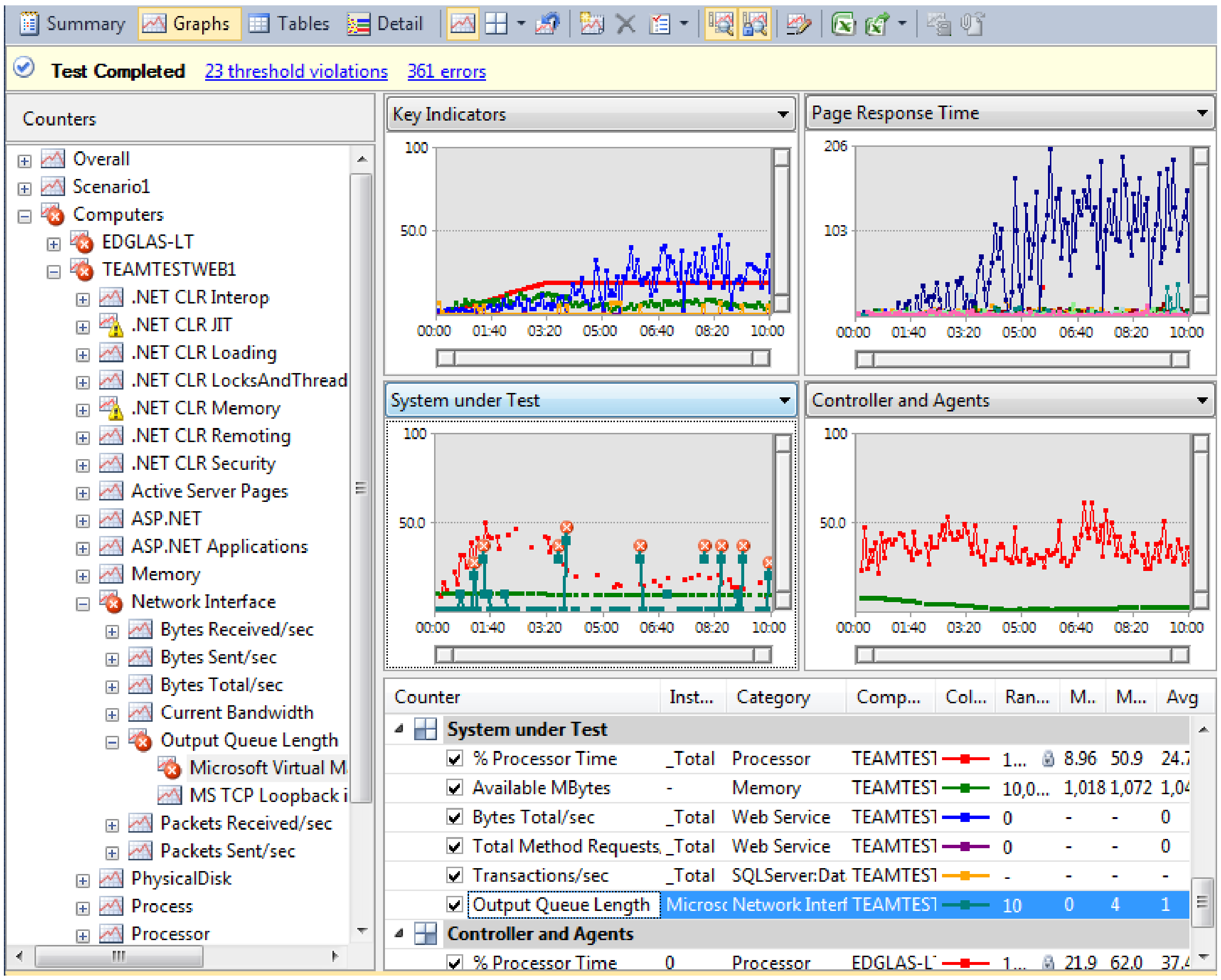
Figure 3 – Select Performance Counter Sets.

You can also extend the testing framework by creating custom data collectors that gather information specific to your needs. Using these capabilities, you can capture custom application logs, network usage, SQL Server calls or virtually any other needed information.

In addition to all of the options open to you – far too many to list here – it scales, scales, and scales. The load testing controllers and agents scale to support true enterprise scenarios with tens of thousands of users at a lower cost than previously available. The best news here is that the licensing is simple – you add additional users through the purchase of user packs. There are no different types of users, additional costs for more controllers, or any hidden surprises. This gives you flexibility and predictability in the cost of your performance testing.

Analyzing Test Results

Load testing is great but if you can't perform an analysis of the test results, the tests themselves are not very useful. The Visual Studio Load Test Analyzer graph view lets you correlate performance slowdowns with activity and conditions on the server, such as errors reported in the event log or excessive resource utilization (Figure 4). Performance Counter thresholds are automatically configured for you in the load test, and allow you to quickly identify resources on the server that are under pressure.



Test Completed 23 threshold violations 361 errors

Counter	Inst...	Category	Comp...	Col...	Ran...		M..	M...	Avg
System under Test									
✓ % Processor Time	_Total	Processor	TEAMTEST	🟥	1...	🔒	8.96	50.9	24.7
✓ Available MBytes	-	Memory	TEAMTEST	🟩	10,0...		1,018	1,072	1,04
✓ Bytes Total/sec	_Total	Web Service	TEAMTEST	🟦	0		-	-	0
✓ Total Method Requests	_Total	Web Service	TEAMTEST	🟪	0		-	-	0
✓ Transactions/sec	_Total	SQLServer:Dat	TEAMTEST	🟧	-		-	-	-
✓ Output Queue Length	Microsc	Network Interf	TEAMTEST	🟦	10		0	4	1
Controller and Agents									
✓ % Processor Time	0	Processor	EDGLAS-L	🟥	1...	🔒	21.9	62.0	37.4

Figure 4 – Load Test Result Summary.

Using the Virtual User Activity Chart, you can determine what individual virtual users were doing during a performance slowdown (Figure 5).

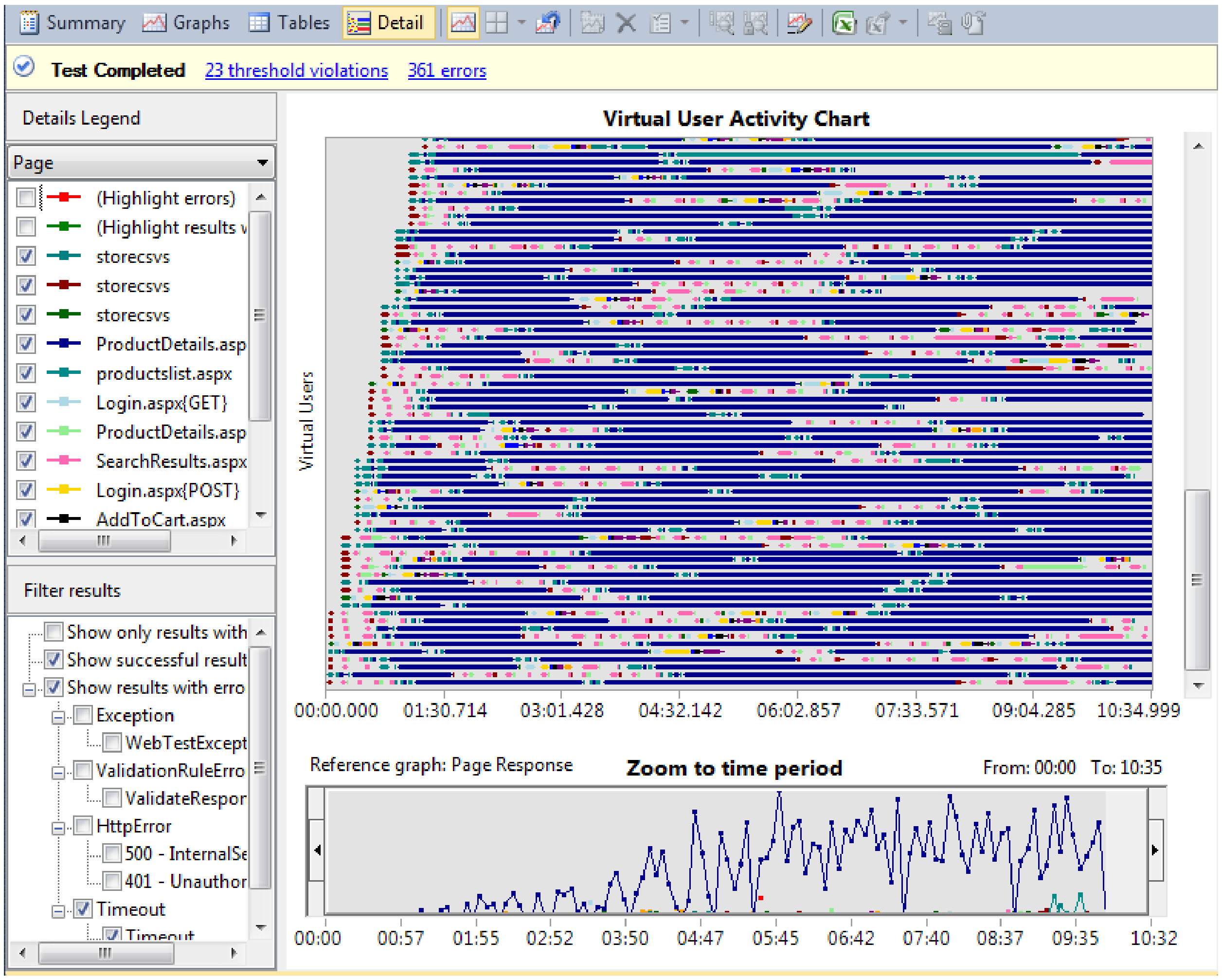


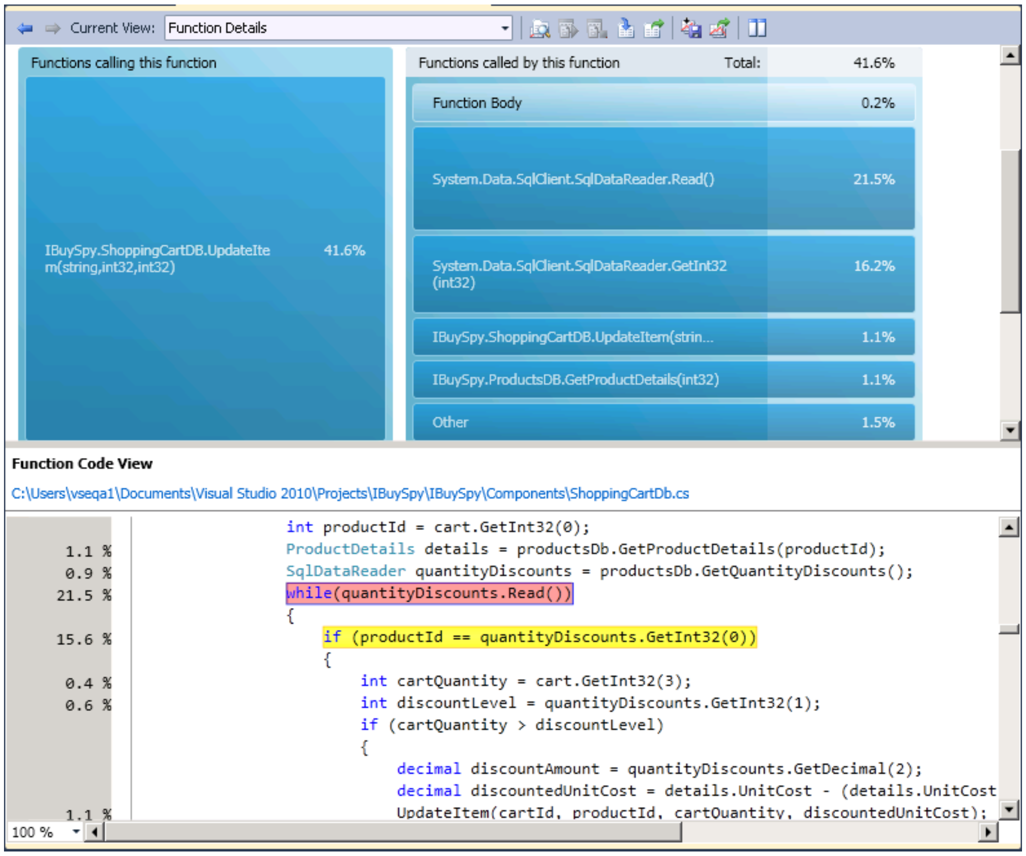
Figure 5 – Virtual User Activity Chart.

FIXING THE BOTTLENECKS

Once you have identified a problem you need to be able to fix it. For this, Visual Studio 2010 uses the Microsoft ASP.NET Profiler data collector. The data collector allows you to sample or instrument your code and even the code in the .NET Framework to analyze it for bottlenecks down to the method level. A typical scenario might involve running a Load test and discovering that one of your scenarios, Order Item, is running slower than it should and you want to discover why this is the case. Once you've identified the problem scenario you can drop down into the ASP.NET profiler to get detailed information on every method called during a session (Figure 6). This lets you identify the hot path which is the slowest path through your code.

What is the ASP.NET Profiler?

This data collector profiles the Internet Information Services (IIS) process. It will monitor the overall performance of IIS relative to a specific application, record performance counters *and* monitor your code down to the method level. You can also gather information on calls made to a database to determine the effect of those calls on your overall application performance.



Functions calling this function	
IBuySpy.ShoppingCartDB.UpdateItem(string,int32,int32)	41.6%

Functions called by this function	Total: 41.6%
Function Body	0.2%
System.Data.SqlClient.SqlDataReader.Read()	21.5%
System.Data.SqlClient.SqlDataReader.GetInt32(int32)	16.2%
IBuySpy.ShoppingCartDB.UpdateItem(strin...	1.1%
IBuySpy.ProductsDB.GetProductDetails(int32)	1.1%
Other	1.5%

Function Code View

C:\Users\vseqa1\Documents\Visual Studio 2010\Projects\IBuySpy\IBuySpy\Components\ShoppingCartDb.cs

```
                int productId = cart.GetInt32(0);
   1.1 %        ProductDetails details = productsDb.GetProductDetails(productId);
   0.9 %        SqlDataReader quantityDiscounts = productsDb.GetQuantityDiscounts();
  21.5 %        while(quantityDiscounts.Read())
                {
  15.6 %            if (productId == quantityDiscounts.GetInt32(0))
                    {
   0.4 %                int cartQuantity = cart.GetInt32(3);
   0.6 %                int discountLevel = quantityDiscounts.GetInt32(1);
                        if (cartQuantity > discountLevel)
                        {
                            decimal discountAmount = quantityDiscounts.GetDecimal(2);
                            decimal discountedUnitCost = details.UnitCost - (details.UnitCost
   1.1 %                    UpdateItem(cartId, productId, cartQuantity, discountedUnitCost);
```

Figure 6 – Function Details view.

Find Performance Problems for Global Customers Before Deployment

The ASP.NET Profiler works well with application performance issues. Have you ever rolled out an application that worked great for local customers, but performed poorly for global customers accessing it over a WAN? Many performance issues are due to chatty clients or clients that send and receive large amounts of data. These applications work fine on a LAN, but have poor performance over a WAN. VS 2010 enables you to run applications while simulating a WAN, enabling you to find these types of performance problems early.

True Network Emulation

Microsoft Visual Studio 2010 uses software-based true network emulation for all test types. This emulation simulates network conditions by direct manipulation of the network packets. This allows for easy simulation of wired or wireless networks and allows for filtering at the packet level.

Easily Create Large Data Sets Using Data Generation

An area that often goes overlooked is the amount of data in the database. The more data there is, the more likely it is for queries to execute slowly and introduce deadlocks. One of the features of Visual Studio Premium and Ultimate is the ability to generate test data. Using this feature you can generate and test large amounts of realistic data with very little effort (Figure 7).

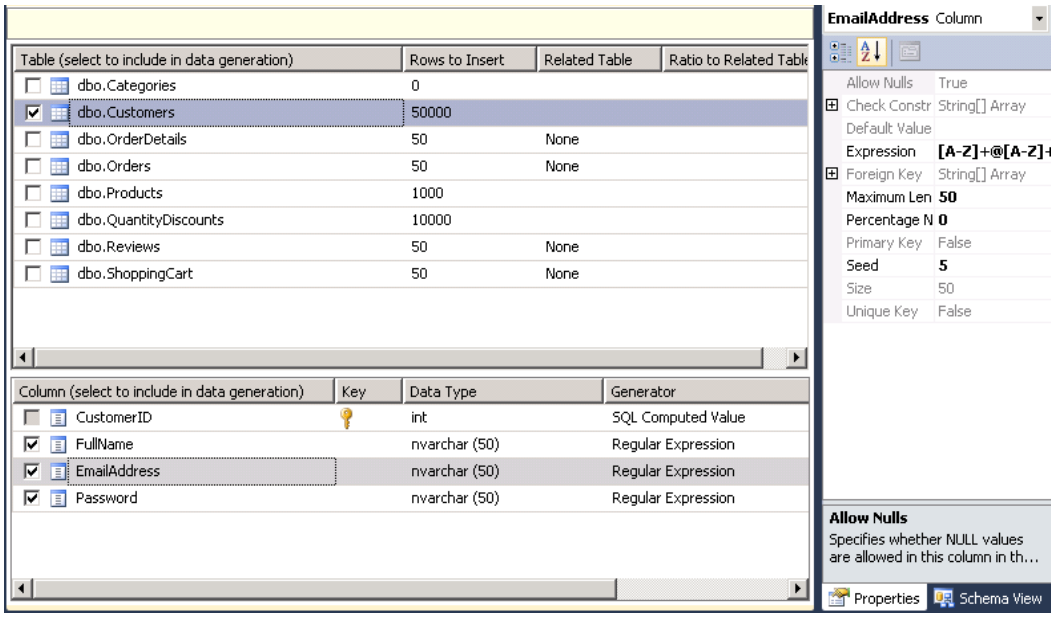


Table (select to include in data generation)	Rows to Insert	Related Table	Ratio to Related Table
☐ dbo.Categories	0		
☑ dbo.Customers	50000		
☐ dbo.OrderDetails	50	None	
☐ dbo.Orders	50	None	
☐ dbo.Products	1000		
☐ dbo.QuantityDiscounts	10000		
☐ dbo.Reviews	50	None	
☐ dbo.ShoppingCart	50	None	

Column (select to include in data generation)	Key	Data Type	Generator
☐ CustomerID	🔑	int	SQL Computed Value
☑ FullName		nvarchar (50)	Regular Expression
☑ EmailAddress		nvarchar (50)	Regular Expression
☑ Password		nvarchar (50)	Regular Expression

EmailAddress Column

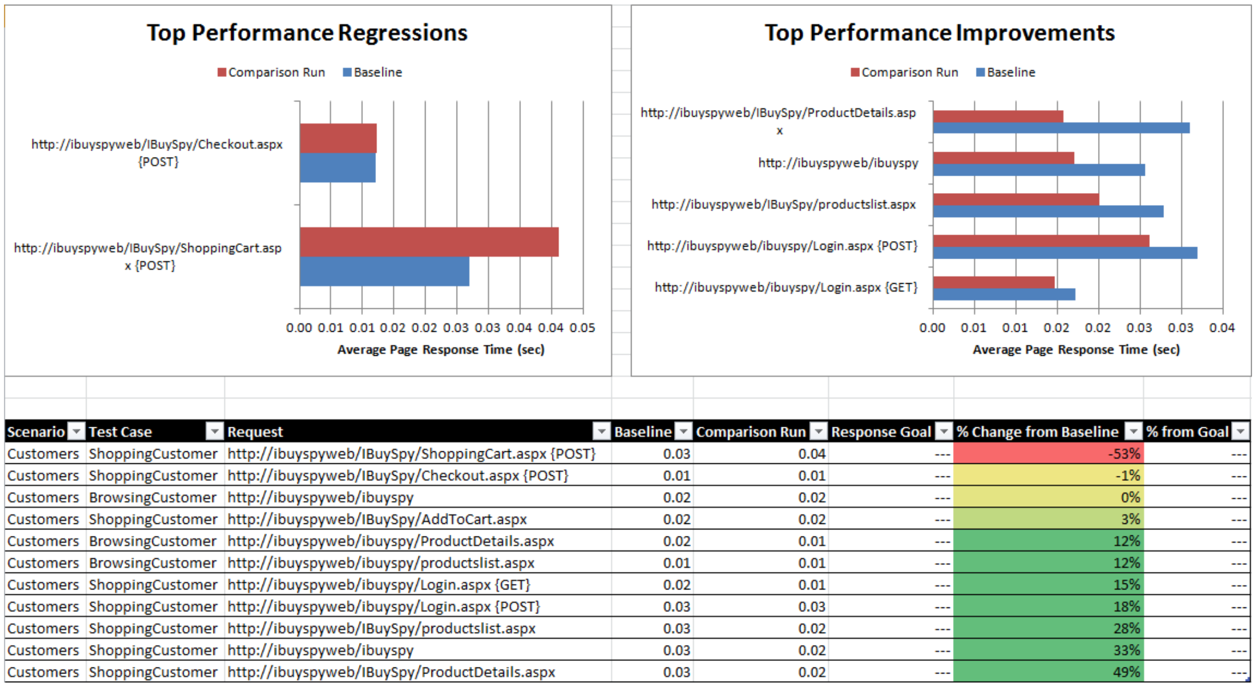
Allow Nulls	True
Check Constr	String[] Array
Default Value	
Expression	**[A-Z]+@[A-Z]+**
Foreign Key	String[] Array
Maximum Len	**50**
Percentage N	**0**
Primary Key	False
Seed	**5**
Size	50
Unique Key	False

Allow Nulls
Specifies whether NULL values are allowed in this column in th…

Figure 7 – Data Generation.

Reporting

Visual Studio 2010 introduces a new set of reports which help you analyze test runs and share results with stakeholders. Using the analytical power of Microsoft Excel you can quickly and easily perform comparisons between test runs as shown in Figure 8. In this view you can see the page response time for a given operation – both where the application became slower and where it improved.

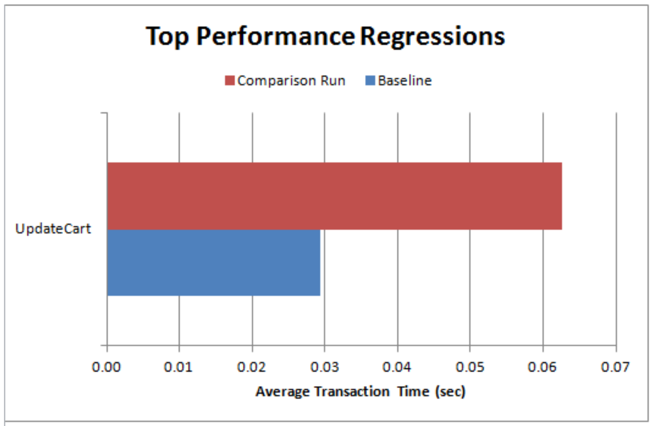


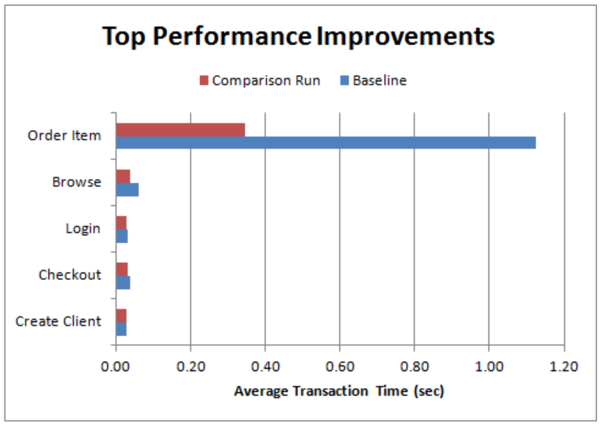
Scenario	Test Case	Request	Baseline	Comparison Run	Response Goal	% Change from Baseline	% from Goal
Customers	ShoppingCustomer	http://ibuyspyweb/IBuySpy/ShoppingCart.aspx {POST}	0.03	0.04	---	-53%	---
Customers	ShoppingCustomer	http://ibuyspyweb/IBuySpy/Checkout.aspx {POST}	0.01	0.01	---	-1%	---
Customers	BrowsingCustomer	http://ibuyspyweb/ibuyspy	0.02	0.02	---	0%	---
Customers	ShoppingCustomer	http://ibuyspyweb/IBuySpy/AddToCart.aspx	0.02	0.02	---	3%	---
Customers	BrowsingCustomer	http://ibuyspyweb/ibuyspy/ProductDetails.aspx	0.02	0.01	---	12%	---
Customers	BrowsingCustomer	http://ibuyspyweb/ibuyspy/productslist.aspx	0.01	0.01	---	12%	---
Customers	ShoppingCustomer	http://ibuyspyweb/ibuyspy/Login.aspx {GET}	0.02	0.01	---	15%	---
Customers	ShoppingCustomer	http://ibuyspyweb/ibuyspy/Login.aspx {POST}	0.03	0.03	---	18%	---
Customers	ShoppingCustomer	http://ibuyspyweb/IBuySpy/productslist.aspx	0.03	0.02	---	28%	---
Customers	ShoppingCustomer	http://ibuyspyweb/ibuyspy	0.03	0.02	---	33%	---
Customers	ShoppingCustomer	http://ibuyspyweb/IBuySpy/ProductDetails.aspx	0.03	0.02	---	49%	---

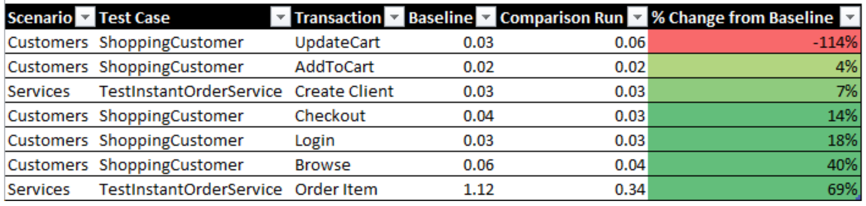
Figure 8 - Page run comparison

Figure 9 shows transaction comparison – that is, how long did it take to perform a set of steps which make up a user scenario? In this figure some of the scenarios are logging onto the system, ordering an item, and checking out.

Performance counter time, errors, overall test results and other reports are also standard reports available and they can be filtered to provide the level of detail that you need. Once the performance bottlenecks have been identified you can drill into the code and make changes as needed to improve performance.







Scenario	Test Case	Transaction	Baseline	Comparison Run	% Change from Baseline
Customers	ShoppingCustomer	UpdateCart	0.03	0.06	-114%
Customers	ShoppingCustomer	AddToCart	0.02	0.02	4%
Services	TestInstantOrderService	Create Client	0.03	0.03	7%
Customers	ShoppingCustomer	Checkout	0.04	0.03	14%
Customers	ShoppingCustomer	Login	0.03	0.03	18%
Customers	ShoppingCustomer	Browse	0.06	0.04	40%
Services	TestInstantOrderService	Order Item	1.12	0.34	69%

Figure 9 - Transaction Times

PERFORMANCE TESTING FOR EVERYONE

The Visual Studio performance testing tools provide a mature, stable, extensible platform for executing, analyzing and acting on test results. Because of the tight integration with the development environment, performance test results are actionable – they don't just make a good looking report. Teams can analyze the data, drill down to the code causing the bottleneck, fix it, and re-run the test to examine the effects of the change. Whether the application has a small number of users or a large number, is mission critical or not, these tools can help you improve the performance of your application without resorting to costly tooling and a steep learning curve.