



# **Addressing Performance Throughout the Life Cycle**

Mike Koza - Subject Matter Expert  
Compuware Corporation

# Agenda

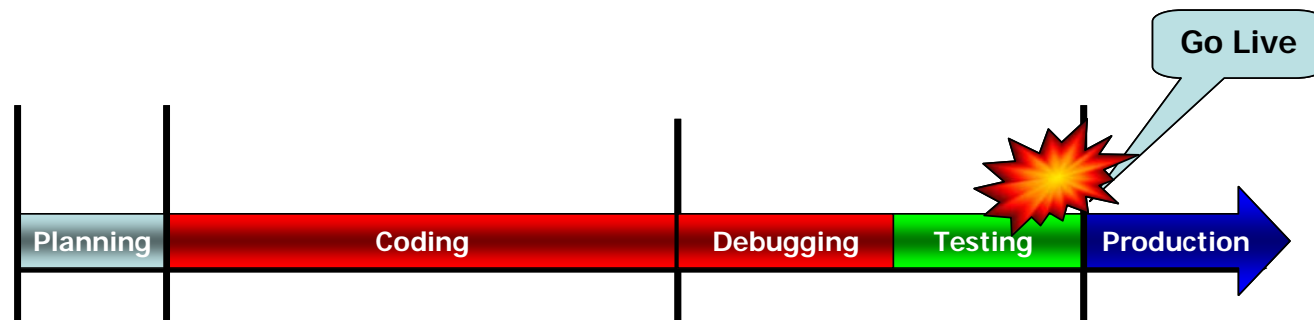
This session will address performance in the following life cycle areas:

- ✓ Requirements Gathering
- ✓ Development
- ✓ Test

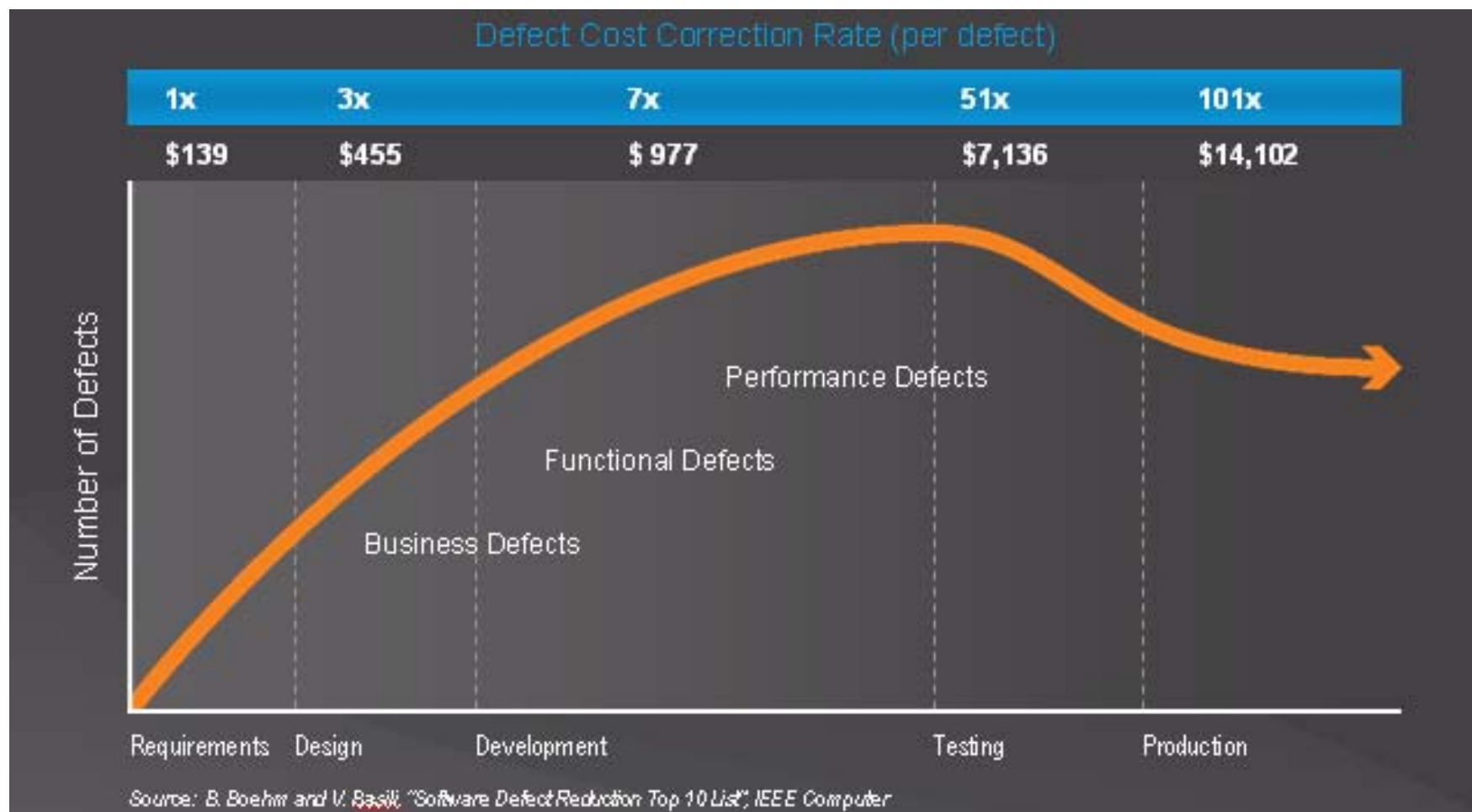
# The IT Challenge

Building quality applications takes **commitment** and **dedication** from the start

- ✓ Traditional QA approach – validation after code-complete
- ✓ With release deadlines fixed, testing is usually cut short
- ✓ Heroic QA efforts are remarkable; they seldom produce what is needed
- ✓ Applications go into production failing to meet the needs of the business
- ✓ Instead, quality must be engineered into the application from the onset

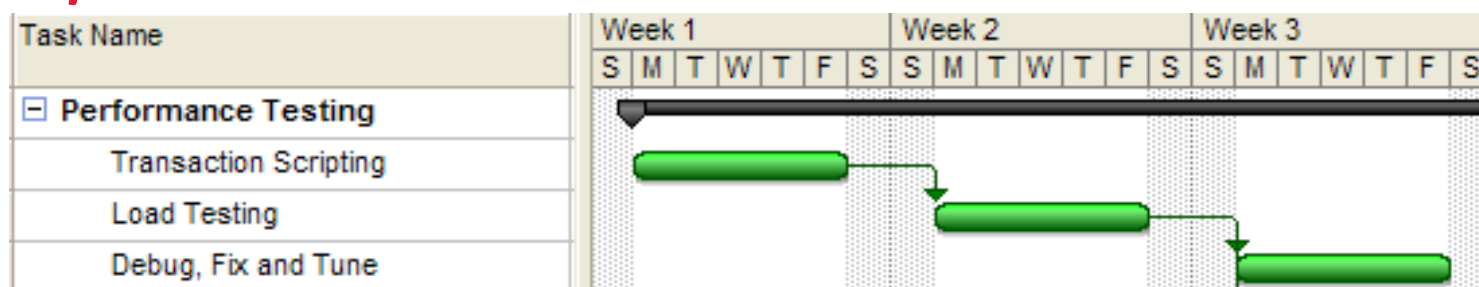


# The IT Challenge

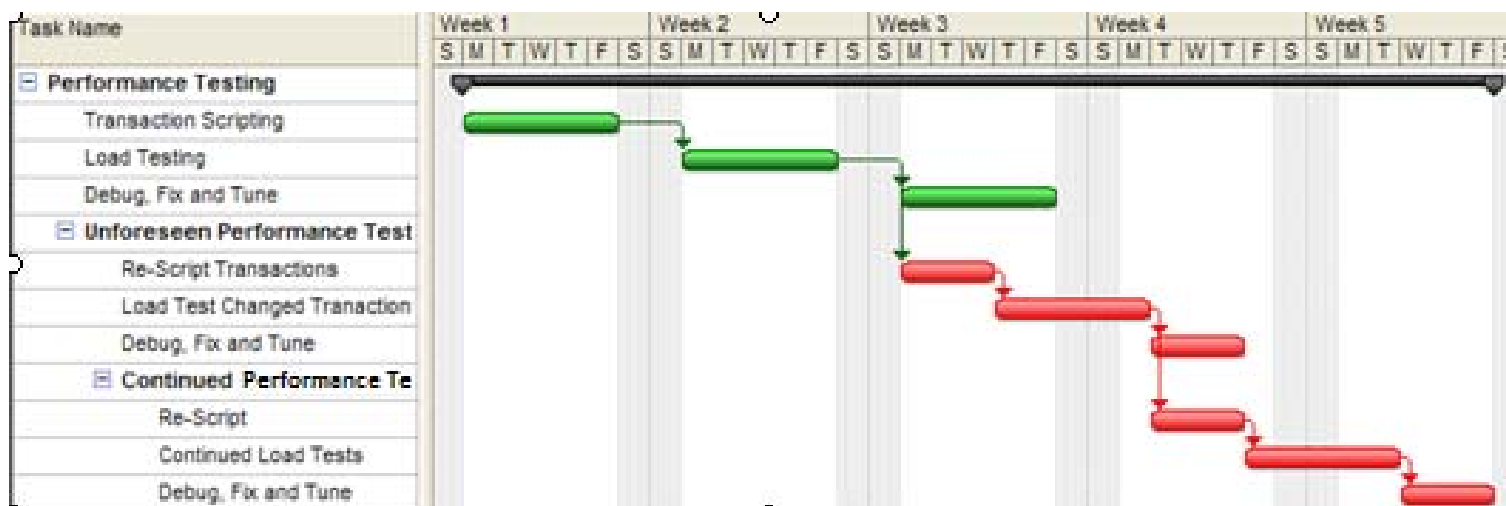


# Traditional Performance Testing Approach

## Expected...



## Actual...



What can be done in  
Requirements Gathering?

# The Solution

Most Requirements focus strictly on application **Functionality**

- ✓ Be sure to capture NON functional requirements
- ✓ Ensure that requirements include key Performance SLAs
- ✓ Identify key business transactions
- ✓ Detail response times for key business transactions
- ✓ Data Requirements

# Identify Critical Business Transactions

Performance Testing is NOT functional Testing!!

Things to think about:

- ✓ Frequently used Transactions
- ✓ Performance Intensive Transactions
- ✓ Business Critical Transactions



# Performance Goals

Performance Goals are difficult to **capture** and **quantify**

- ✓ Get into habit of capturing performance goals early
- ✓ Try capturing performance goals in a **subjective** way first
  - ✓ *Example*: Not any slower than release 10.1
- ✓ Then **quantify** the performance goal
  - ✓ *Example*: Baseline the release to be compared against previous release or competition
- ✓ Use actual users and a prototype to quantify key business transactions

# Identify and Develop Test Data Early

Performance Tests Require Large amounts of data

Things to think about:

- ✓ Uniqueness of Test Data
- ✓ Volume of Test Data
- ✓ Source
- ✓ Sterilization Required?
- ✓ Testing can begin with early builds of the application

# Document User Transaction Mix

Transaction	Starting virtual users	Injection rate per 1 virtual user	Target virtual users
Enrollment	1	12 seconds	10
Signon	1	3 Minutes	150
View Account Summary	1	3 Minutes	10
View Account Details	1	2 minutes 18 seconds	225
Intra FI Transfer	1	12 seconds	145
View Check Images	1	55 seconds	33
Copy Check	1	7 minutes 30 seconds	4
Bill Payment Transaction	1	3 Minutes	10
Check Reorder	1	8 seconds	13

# Performance Requirements

## Response Time SLA

- ✓ Login less than **10 seconds**
- ✓ All other page response times less than **8 seconds**  
*Break down overall transactions into smaller pieces*
- ✓ Overall Transaction response time less than **20 seconds**
- ✓ Based on broadband bandwidth

## Application Concurrency SLA

- ✓ **600** concurrent end users
- ✓ **Server Resource Utilization SLA**
- ✓ Less than **50%** measured as CPU, Memory, Network Utilization and Disk I/O

# Workshop #1 Instructor Guided

## On Line Bookstore Performance Requirements

What can be done in  
Development?

# The Solution

Build processes in development to **test early and often**

- ✓ Don't wait until code is passed over to QA; create tests in dev
- ✓ More testing cycles lead to higher quality better performing code
- ✓ Mandatory Code Reviews
- ✓ Test assets begin to grow in development, then hand off to QA
- ✓ Reduce overall cost of finding and fixing defects early in SDLC



# Code Reviews

Not checking in defects improves quality and performance

“Formal design and code inspections average about 65% in defect removal efficiency.”

“Software Quality: Analysis and Guidelines for Success”  
Caper Jones

“Peer reviews of software will catch 60% of defects.”

Institute of Electrical and Electronics Engineers

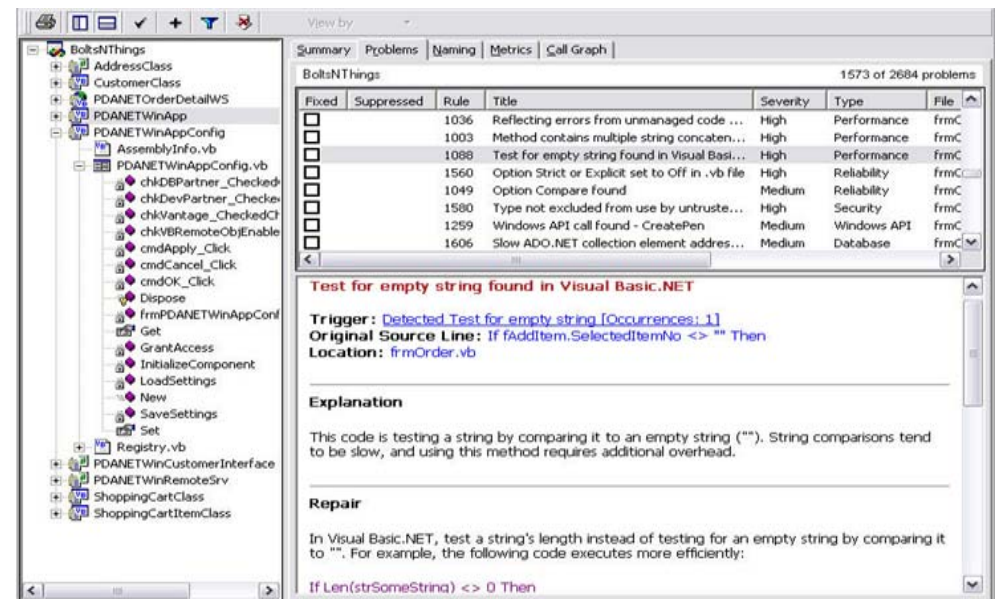
- ✓ Industry data suggests that code reviews are very effective in removing defects
- ✓ My experience shows few development teams perform Code Reviews



# Code Reviews

## Types

- ✓ E-mail pass around reviews
- ✓ Over-the-shoulder reviews
- ✓ Tool-assisted reviews
- ✓ Formal inspection
- ✓ Pair programming



**Source:** *Best Kept Secrets of Peer Code Review, Jason Cohen*

# Code Reviews

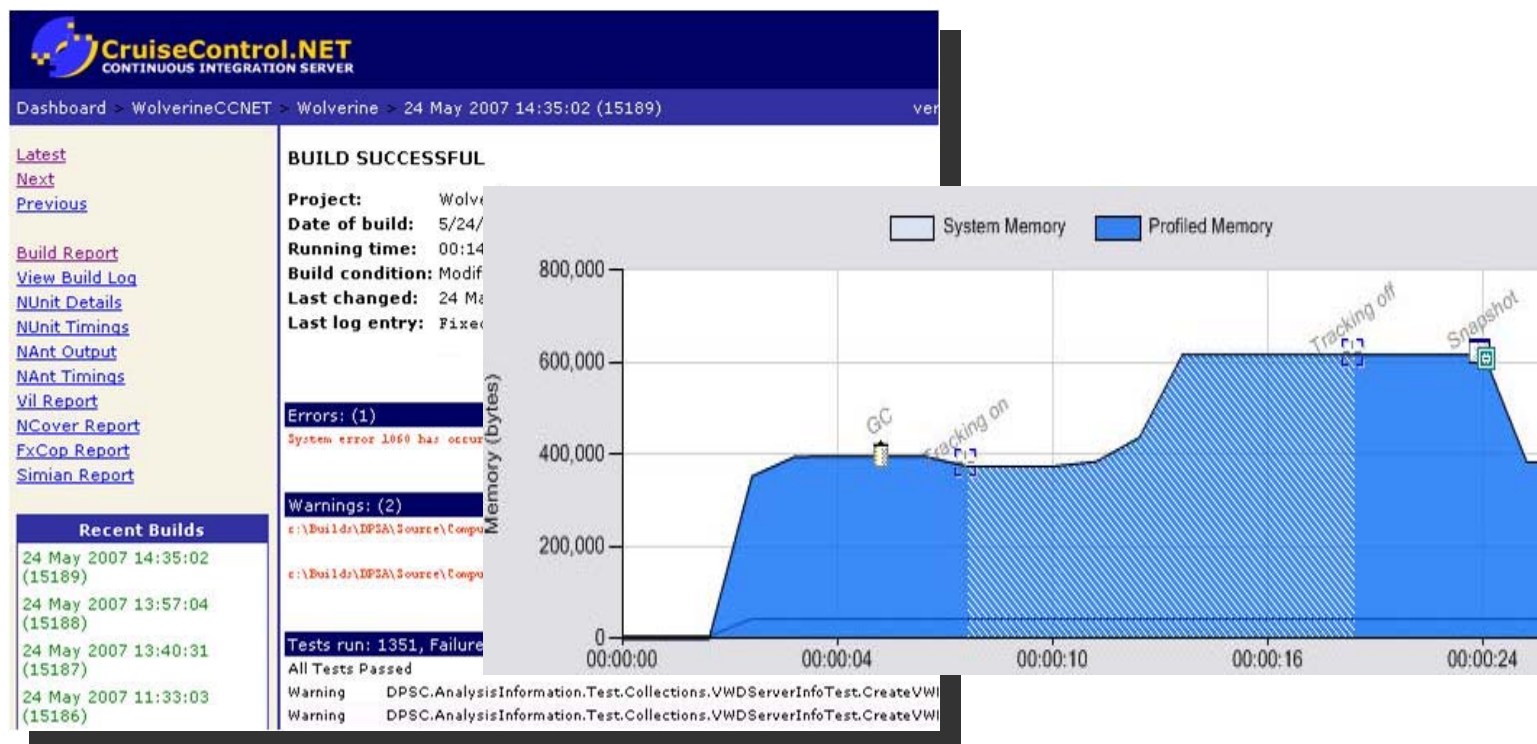
## Best practices

### **Make peer code review mandatory**

- ✓ Note who reviewed the code in check-in comments
- ✓ Knowing a peer is going to review all check-ins forces developers to write better code
- ✓ Explaining and walking through code helps developers understand their code better
- ✓ Helps cross-train team members in different components

# Begin to Understand Performance Issues in Development

- ✓ Consider using Profilers to understand the impact of memory, CPU, and wait time during application development
- ✓ Problems are identified as they are introduced, instead of being found in QA



# Code Coverage

**How well have I tested my application?**

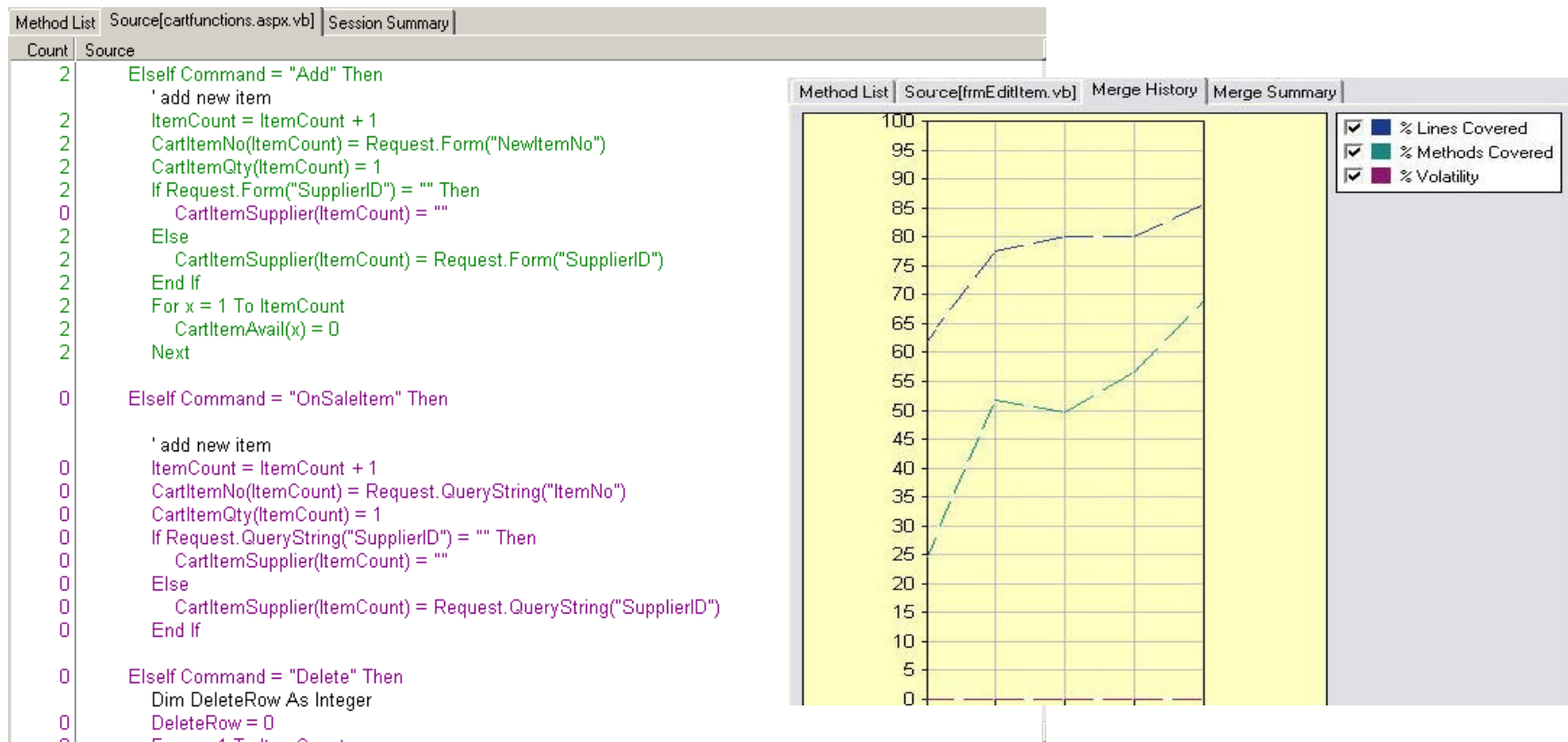
**Am I willing to assume the risk of un-tested code?**

- ✓ Performance Issues LOVE to lurk in untested code!
- ✓ Pinpoint the portions of an application left unexecuted during testing
- ✓ Exclude areas of non-concern (i.e. testing frameworks)

<div> <div></div> 1335 of 2530 lines executed (52.767%) </div> <div> <div></div> 207 of 400 methods called (51.750%) </div>					
Method List   Source[shippingmethod.aspx.vb]   Session Summary					
Method Name	% Covered	Called	# Lines Not Executed	# Lines Executed	
BNTNETWebApp.products.Page_Load(ByVal sender As Object, ByVal e As Ev...	98.361	1	1	60	
BNTNETWebApp.onsale.Page_Load(ByVal sender As Object, ByVal e As Even...	93.478	6	3	43	
BNTNETWebApp.payment.Page_Load(ByVal sender As Object, ByVal e As Ev...	93.333	2	4	56	
BNTNETWebApp.account.Page_Load(ByVal sender As Object, ByVal e As Eve...	93.023	1	3	40	
BNTNETCalcShippingWS.CalcShippingWS.getShipperInfoByIndex(Int32)	90.698	10	4	39	
BNTNETCalcShippingWS.CalcShippingWS.getCheapShipper(void)	89.655	10	3	26	
BNTNETWebApp.login.Page_Load(ByVal sender As Object, ByVal e As Event...	86.667	3	4	26	
BNTNETWebApp.Global.session_start(void)	84.706	1	13	72	
BNTNETWebApp.itemlink.Page_Load(ByVal sender As Object, ByVal e As Eve...	80.000	2	1	4	
BNTNETWebApp.checklogin.Page_Load(ByVal sender As Object, ByVal e As E...	78.125	3	7	25	
BNTNETCalcShippingWS.CalcShippingWS.getTotShipping(Int32, Int32)	69.767	2	13	30	
BNTNETWebApp.cart.Page_Load(ByVal sender As Object, ByVal e As EventA...	66.465	2	111	220	
BNTNETCalcShippingWS.CalcShippingWS.Dispose(Boolean)	57.143	24	3	4	
BNTNETCalcShippingWS.CalcShippingWS.getShipperCount(void)	53.571	2	13	15	
BNTNETWebApp.cartfunctions.Page_Load(ByVal sender As Object, ByVal e A...	51.316	4	37	39	
BNTNETWebApp.itemdetail.InitializeComponent(void)	0.000	0	2	0	
BNTNETWebApp.itemdetail.Page_Init(ByVal sender As Object, ByVal e As Ev...	0.000	0	3	0	
BNTNETWebApp.itemdetail.Page_Load(ByVal sender As Object, ByVal e As E...	0.000	0	95	0	

# Code Coverage

- ✓ Merge sessions to present a clear picture of testing progress over time
- ✓ Discover stability of code base
- ✓ Ensure areas that have been changed have been tested as well





# The Most Difficult Step

## Reporting quality status

- ✓ Many profiling tools allow you to export the data they collect
- ✓ Combine the data that is most important to your organization into a report

### Test Coverage - Summary of tested code, test gaps and code volatility

Project	C:\Program Files\Compuware\BNTNET\BNTNETWinApp\bin\BNTNETWinApp.exe		
Session date	10/23/2008 12:06:42 PM		
Total Line Coverage	3881 of 6060 lines.	64%	
Total Method Coverage	171 of 362 methods.	47%	
Volatility	0%		
Stability	100%		

 Details

### Code Review - Summary of code compliance, maintainability and complexity

Reviewed	Friday, February 06, 2009 12:14 PM			
	3	classes		
	109	methods		
	6049	lines of code		
	438	errors detected		
		180 High	258 Medium	0 Low
				0 Warnings
		Highest complexity recorded:		26 - Medium

 Details

# Case Study – Insurance Company

# Current Situation

This insurance company was enhancing a large, mission-critical application

- ✓ Adding new functionality and re-architecting a very stable and reliable legacy system
- ✓ First release missed initial release date
- ✓ Once deployed, this release contained many quality problems



# Solution and Results

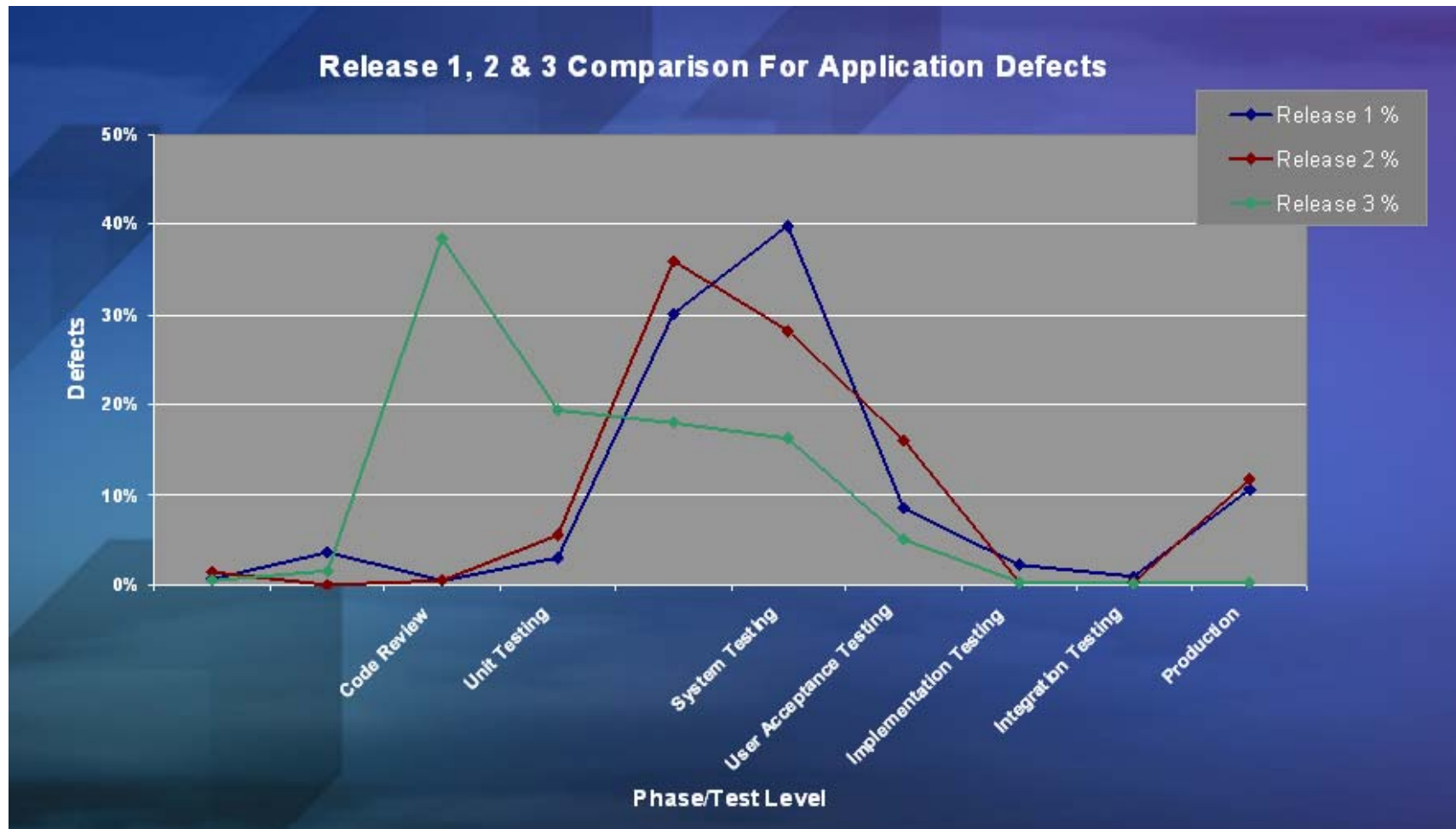
## Solution was deployed in a phased approach

- ✓ Unit and functional tests were captured on a daily basis as code was developed
- ✓ Automated build ran nightly
- ✓ Defects were reported to development for next day fix

## Positive results were realized on next release

- ✓ Next release was deployed on schedule with minimal defects
- ✓ Estimated savings of \$2M to \$8M in avoided rework and support costs

# Solution and Results



# What can be done in Test?

# Why Load Testing ALONE is Not Enough

## NOT ENOUGH INFORMATION

- ✓ Only delivers general response time, throughput or server metrics
- ✓ Does not identify where bottlenecks are, across environment or inside application
- ✓ Doesn't get to the root cause of the problem
- ✓ Leads to finger-pointing

## NOT TIMELY

- ✓ You have to wait until after load testing to understand whether you have a problem

## RESULT...

- ✓ Missed delivery dates
- ✓ Poor-quality applications
- ✓ High-end resources involved in resolving problems and waiting until the end of testing
- ✓ Costly/unnecessary infrastructure changes to fix problems

# What makes for a better Load Test?

## **PREDICTION:**

- ✓ Are you ready to Load Test
- ✓ Predict performance under varying conditions
- ✓ Identify impact of network on application from multiple locations
- ✓ Pinpoint bottlenecks across application tiers
- ✓ Fix code prior to conducting load testing

## **TROUBLESHOOTING:**

- ✓ Deeper analysis during load test
- ✓ Pinpoint application performance and memory issues DURING the Load Test
- ✓ Perform fewer application retests

# Case Study - Online Banking

# Setting the scene

- ✓ Online banking arm of large corporate finance house
- ✓ Urgent requirement to validate existing infrastructure capacity and to investigate capacity to handle further growth
- ✓ Limited time to execute

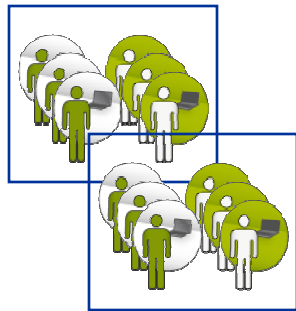
# Performance Goals

- ✓ Response Time SLA
  - ✓ Login less than 10 seconds
  - ✓ All other page response times less than 8 seconds
  - ✓ All transaction response times less than 20 seconds
  - ✓ Based on broadband bandwidth
- ✓ Concurrency SLA
  - ✓ Support 600 concurrent end users
- ✓ Server Utilization SLA
  - ✓ Server utilization < 50% measured as CPU, Memory and Disk I/O

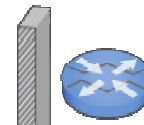
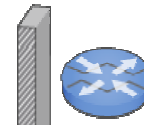
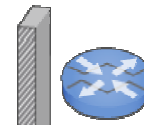
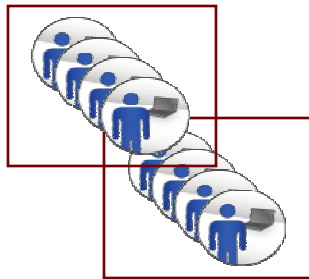


# Load testing alone won't identify hidden problems

External Users



Internal Users



Bank Data Center

Web Servers

Insufficient Capacity

App Servers

Slow Methods

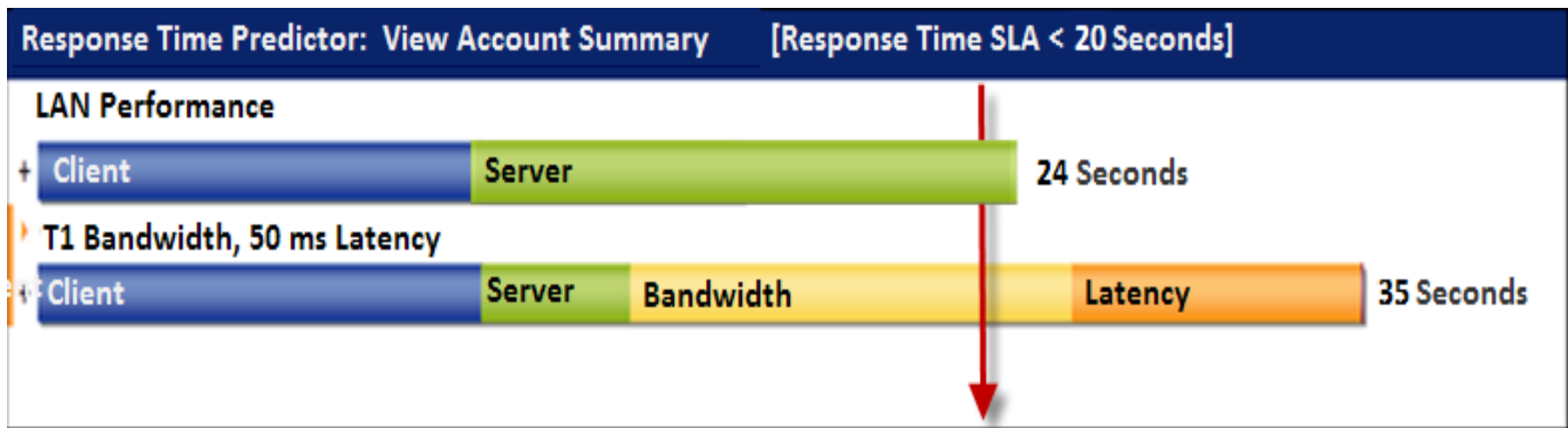
DB Servers

Contention Issues

Bad SQL

# Profile – Predicting WAN sensitivity

Increase in response time of 11 seconds when connecting over T1 link with 50ms latency

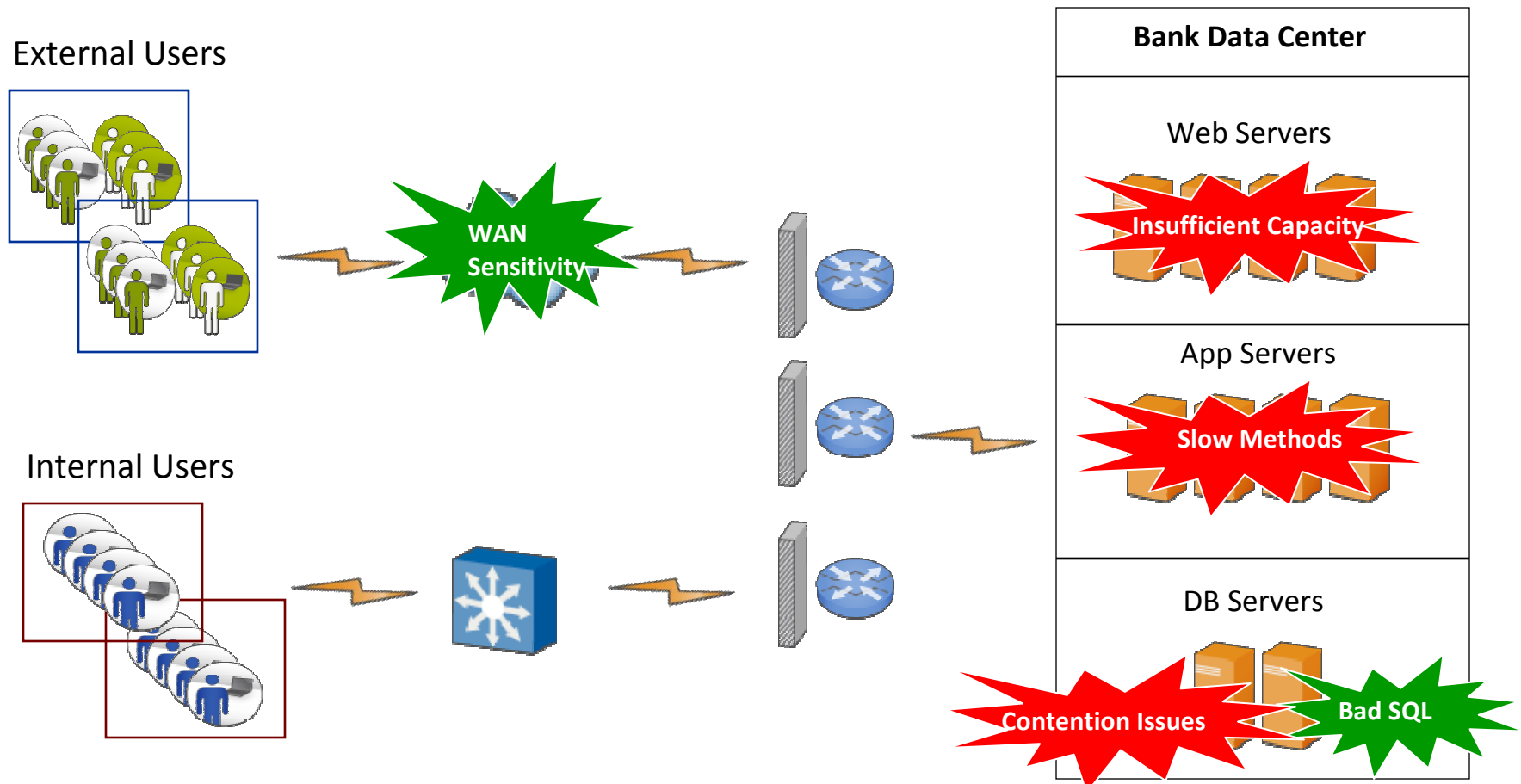


# Profile – Bad SQL Performance

SQL call taking in excess of 13 seconds to complete

	Name	Client	Server	Start Time	Duration	Network Transmission Time	Seconds
256	ORACLE "Cancel current operation"	GIOS Client 54863	Server 1735	101.914350895	0.009628929	0.000117120	
257	ORACLE "SELECT acc_ext_reference, to_char(acc_cldn_acct_id) FROM cp_v_	GIOS Client 54863	Server 1735	101.924817615	3.522894730	0.000076000	
258	ORACLE "Cancel current operation"	GIOS Client 54863	Server 1735	105.42088268	0.170042579	0.000116960	
259	ORACLE "select NULL & description, NULL & target_id, NULL & account_n@o	GIOS Client 54863	Server 1735	105.620954778	0.067022499	0.000191040	
260	ORACLE "SELECT acc_ext_reference, to_char(acc_cldn_acct_id) FROM cp_v_	GIOS Client 54863	Server 1735	105.689371778	13.770325390	0.000358080	
261	ORACLE "ROLLBACK"	GIOS Client 54863	Server 1735	119.463467552	2.792562774	0.000274320	
262	ORACLE "Cancel current operation"	GIOS Client 54863	Server 1735	122.256196300	0.011284484	0.000089520	
263	ORACLE "Cancel current operation"	GIOS Client 54863	Server 1735	122.267595610	0.035048097	0.000166480	
264	ORACLE "Cancel current operation"	GIOS Client 54863	Server 1735	122.302786566	0.432089339	0.000104080	
265	ORACLE "Cancel current operation"	GIOS Client 54863	Server 1735	122.735023522	6.721173175	0.001576160	
266	ORACLE "Cancel current operation"	GIOS Client 54863	Server 1735	129.456343959	0.014735627	0.000089520	
267	ORACLE "Cancel current operation"	GIOS Client 54863	Server 1735	129.471191683	0.052413790	0.000291200	
268	ORACLE "Cancel current operation"	GIOS Client 54863	Server 1735	129.523748090	0.450526652	0.000104080	
269	ORACLE "Cancel current operation"	GIOS Client 54863	Server 1735	129.974423392	0.569159598	0.000249280	

# Profiling Identifies hidden problems **BEFORE** the Load Test

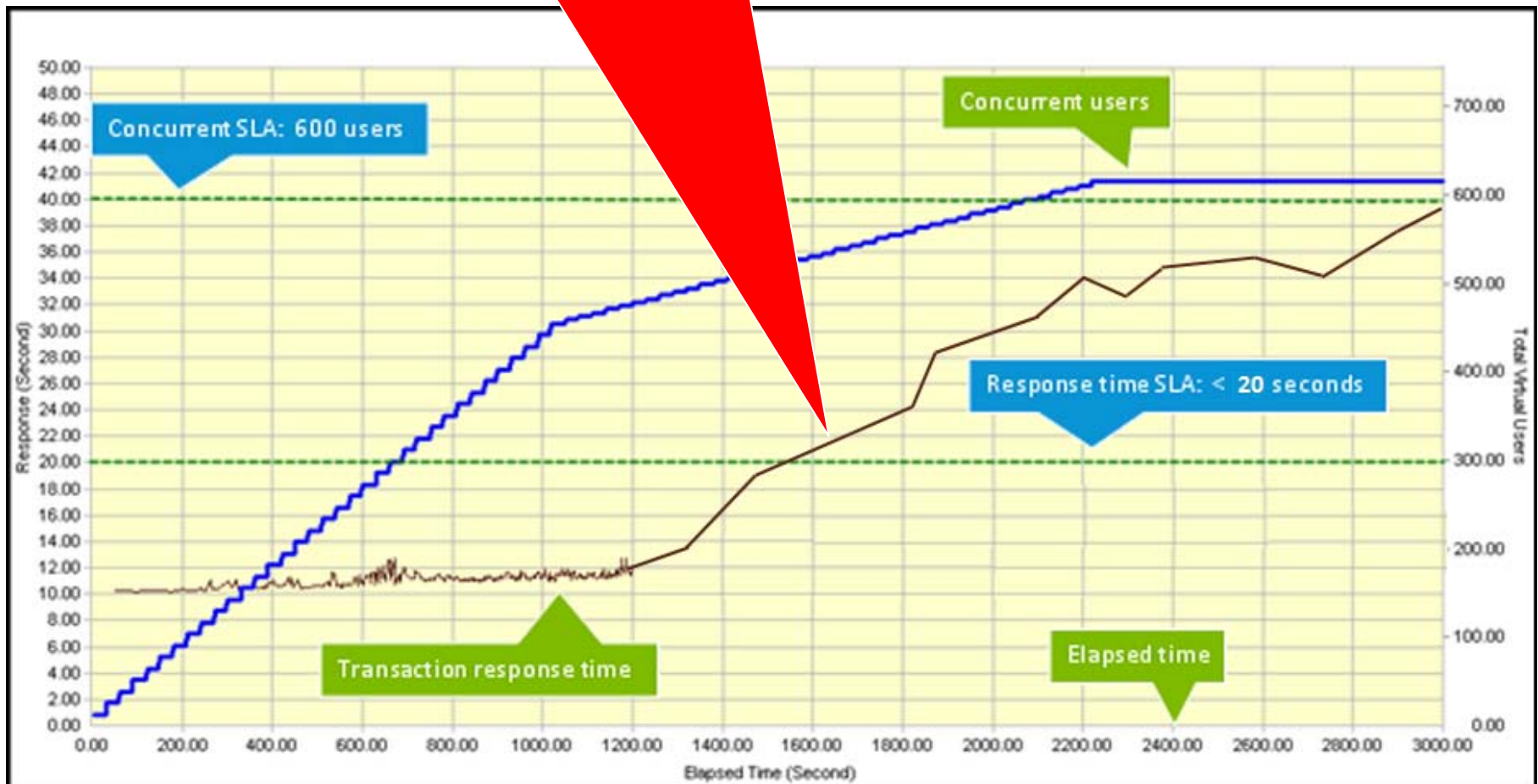


# Load Testing – Transaction performance

Transaction performance exceeds response time SLA

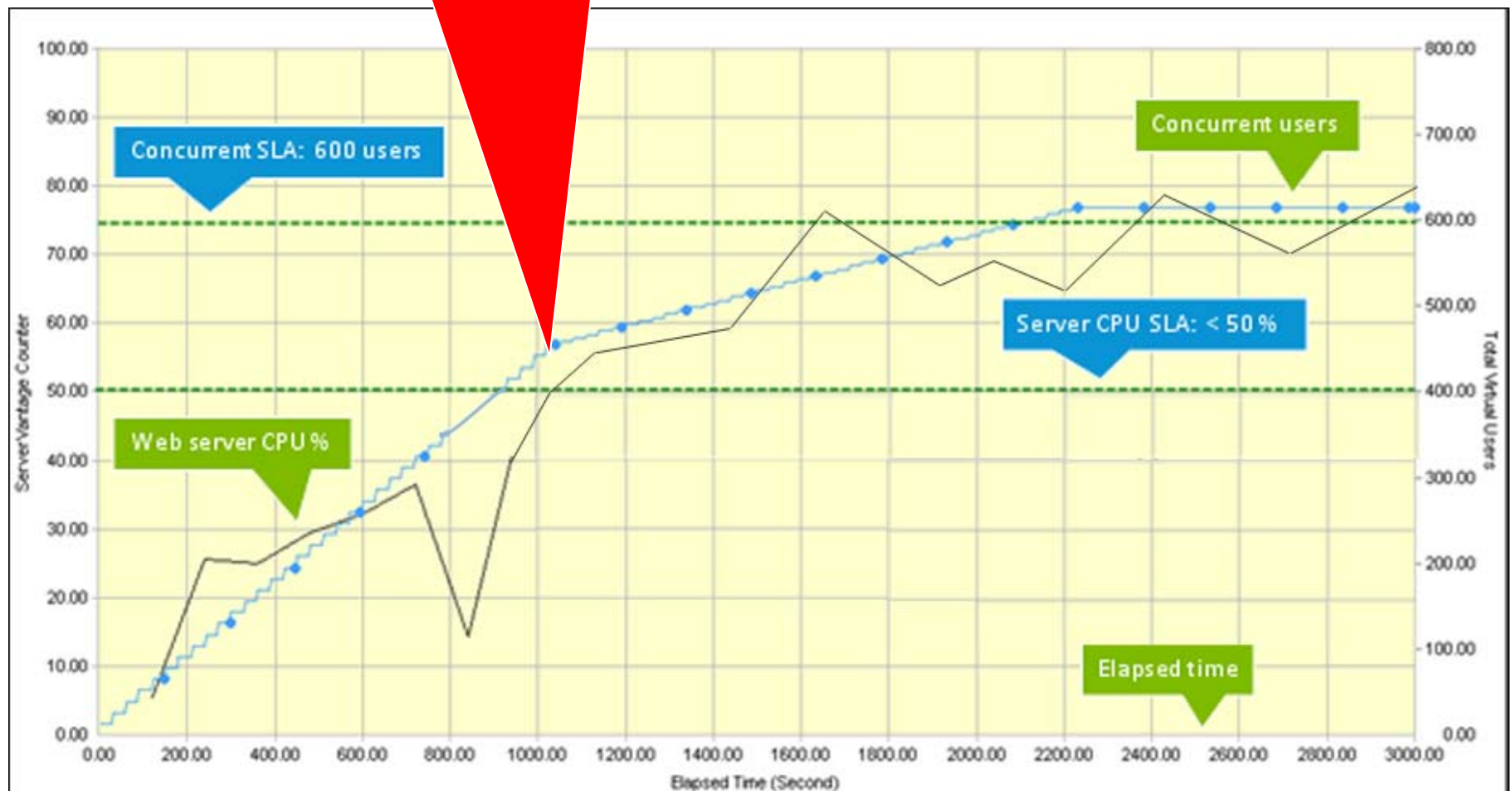
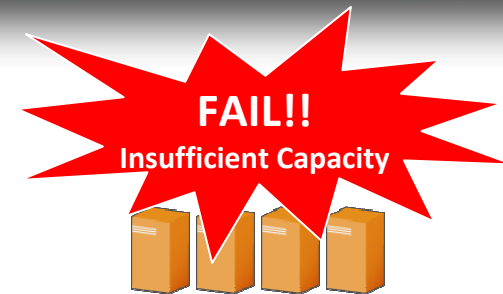
**FAIL!!**

Response Time



# Load Testing – Server performance

Web server CPU utilization breaches SLA





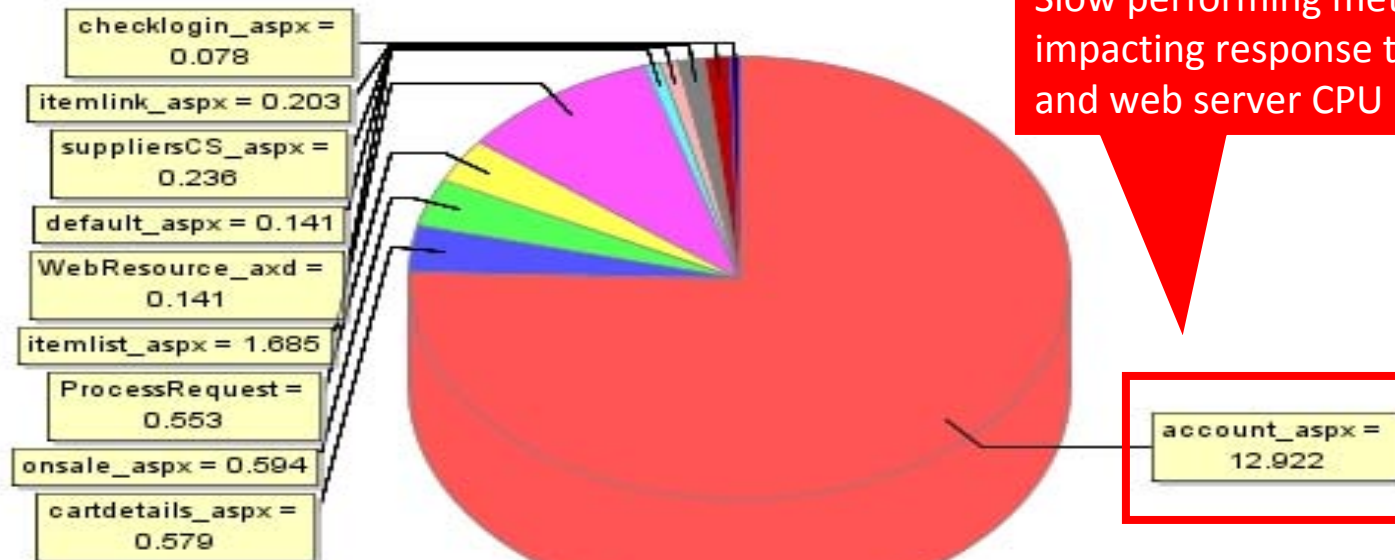
# Load Testing – Inside the Application

Analysis inside the JVM and CLR

**FAIL!!**  
Slow Method



**TopMethods**  
(in seconds)



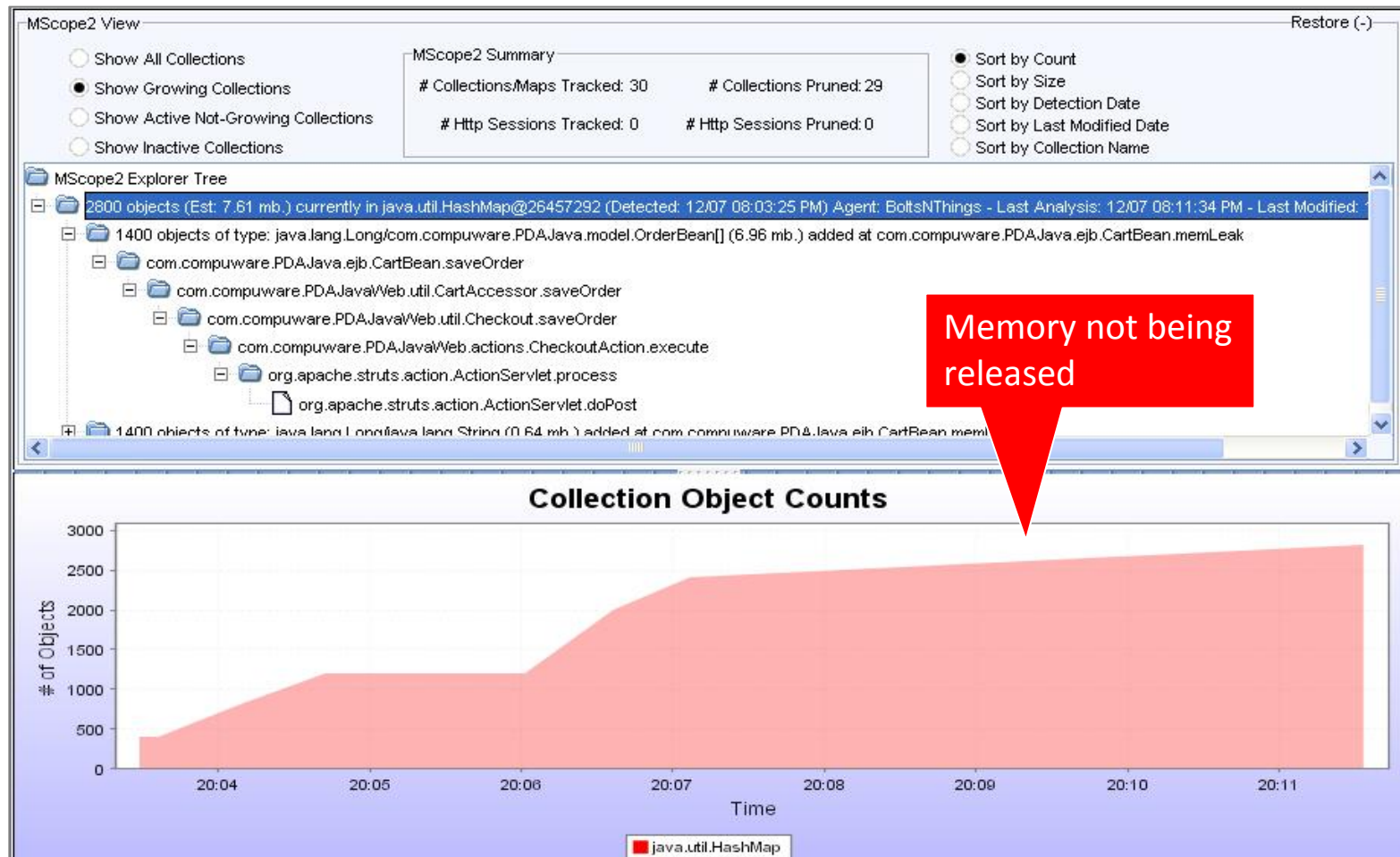
Slow performing method  
impacting response time  
and web server CPU

account\_aspx = 12.922   cartdetails\_aspx = 0.579   onsale\_aspx = 0.594   ProcessRequest = 0.553   itemlist\_aspx = 1.685  
WebResource\_axd = 0.141   default\_aspx = 0.141   suppliersCS\_aspx = 0.236   itemlink\_aspx = 0.203   checklogin\_aspx = 0.078

# Load Testing – Inside the Application

Analysis inside the JVM and CLR

**FAIL!!**  
Memory Leak



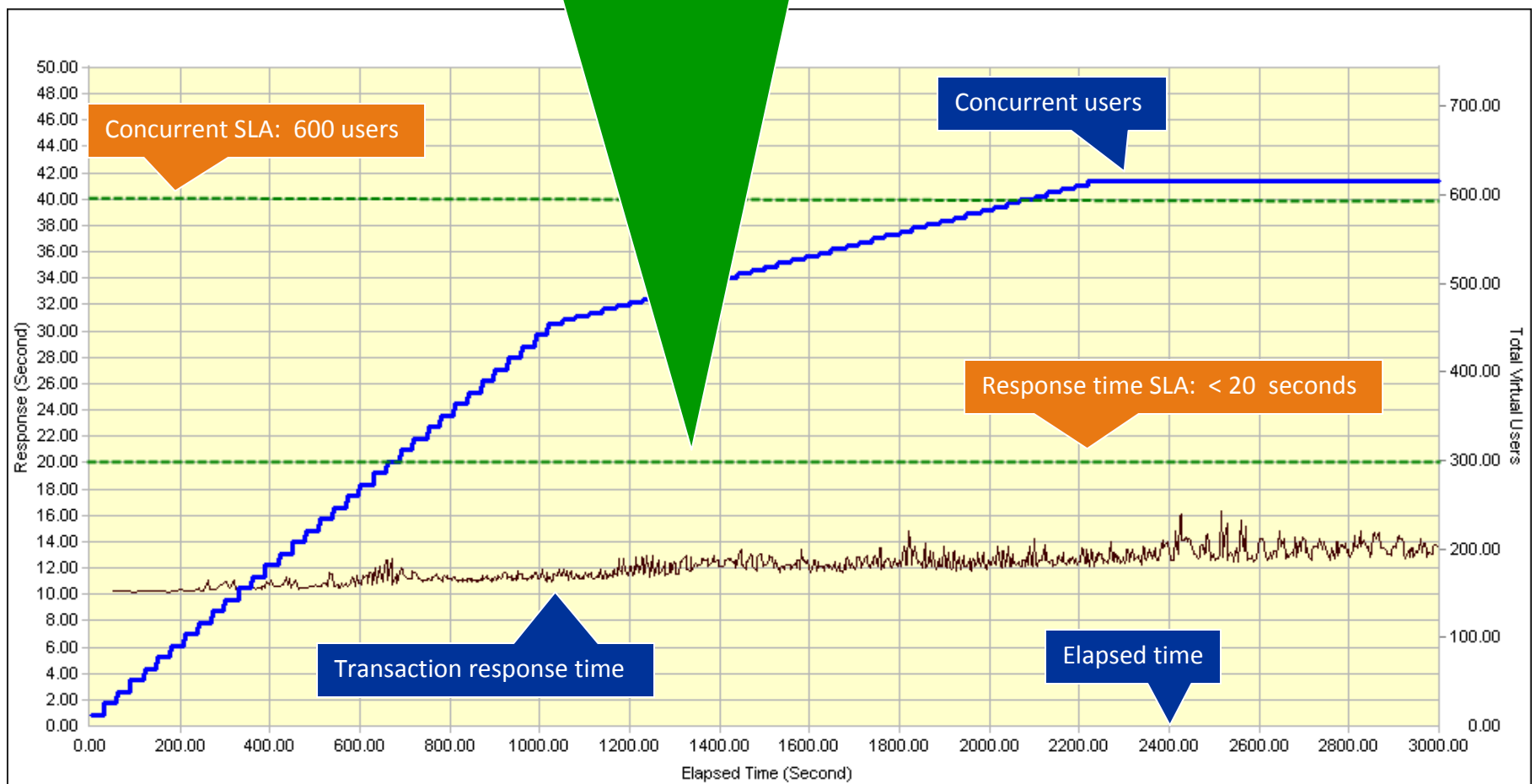


# Load Testing – Transaction performance

**SUCCESS!!**



Transaction performance remains below response time SLA

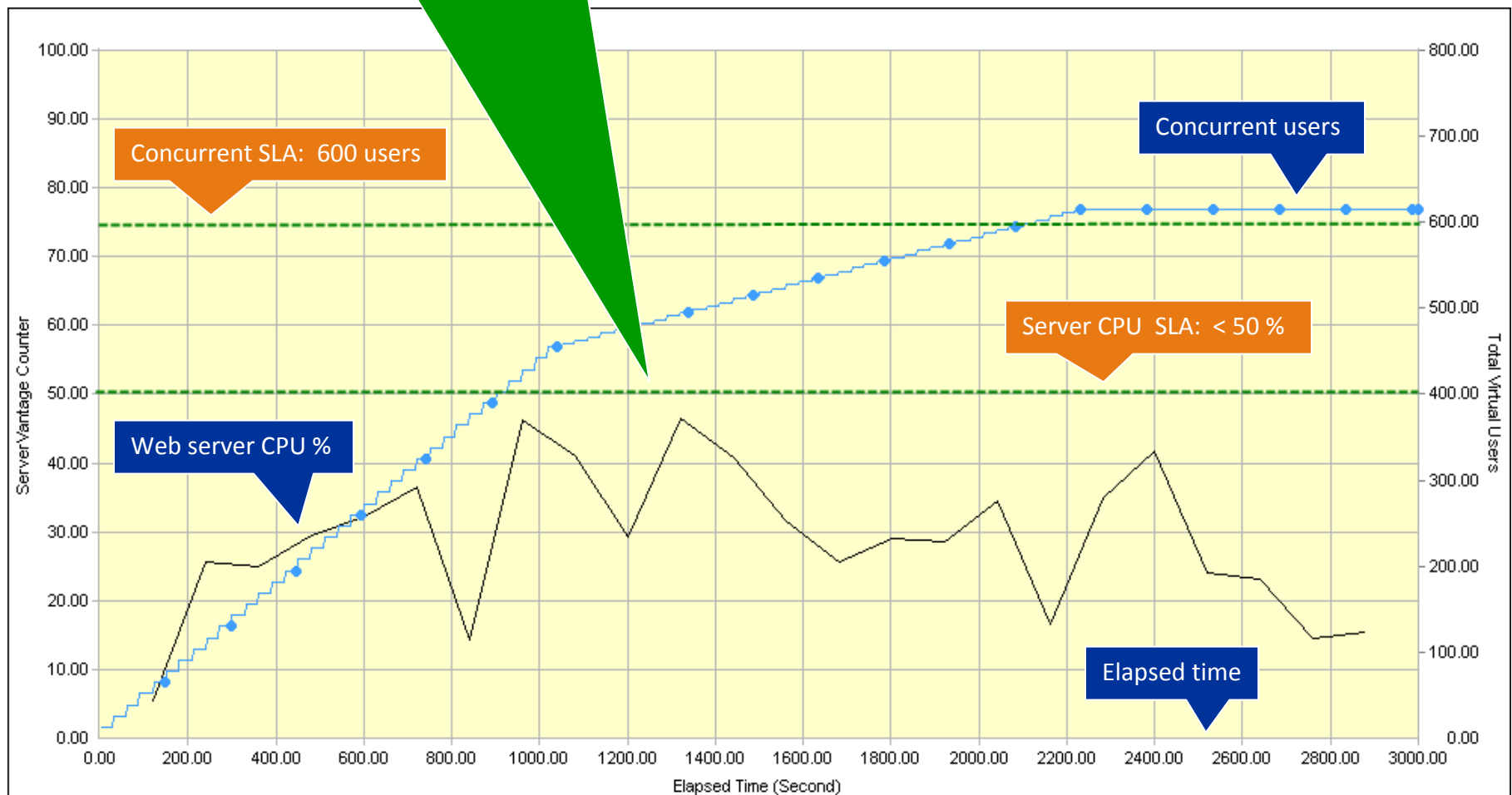


# Load Testing – Server performance

**SUCCESS!!**

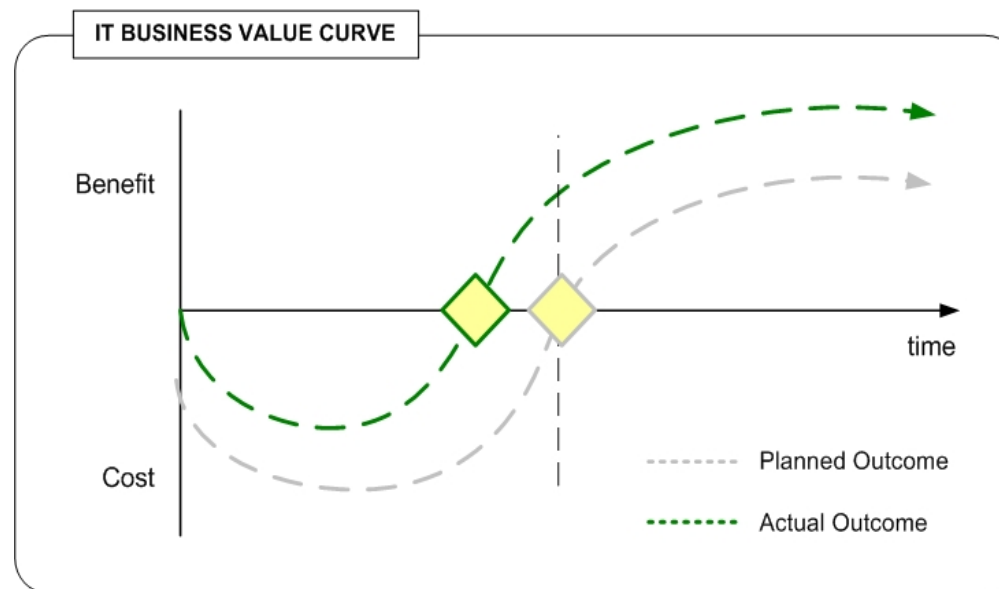


Web server CPU utilization remains below SLA



# Summary

- ✓ Testing Early ,Often and Automatically allows IT to **build quality into** the application from the earliest phases of the development life cycle, rather than attempting to **test** it in after the fact
- ✓ This approach to finding defects early allows the business to realize value from the application from the time it is put into production



# Q&A