Lifecycle diagnostics for .NET applications: identify and resolve performance bottlenecks faster

White paper

Table of contents

Executive summary

Organizations are realizing that business performance is directly linked to application performance. When applications are unavailable or perform poorly, your employees are less productive, your customers complain, you miss service level agreements (SLAs) and you can lose sales. Yet many companies continue to struggle with implementing the right tools and processes to properly test their applications before, during and after the "go-live" rollout—particularly applications developed using the rich yet complex, multi-tiered Microsoft® .NET environment.

How can your architects and developers determine whether applications can scale as expected under load conditions? How do your quality assurance (QA) engineers know they are finding problems before deployment? What do your operations teams communicate to your developers when they identify performance problems?

All too often, organizations consider critical functions such as performance testing as discrete tasks, not as an integral part of the application lifecycle. When release schedules tighten, QA responds to increasing time pressure and "throws" applications into production. When problems arise, there is usually little or no collaboration among architects, development teams, QA testing teams and operations teams. The results often include:

- An inability to quickly isolate and resolve performance problems that are identified in production, staging, load testing or development
- Added expense and delays in moving applications into production
- A lack of proper manageability and quick problem diagnosis for applications being monitored in production
- Applications that fail to meet end users' expectations and your business requirements for performance and availability

You need a more holistic approach for managing your .NET application lifecycle. You need an approach that plans for testing throughout pre-production and post-production processes, that encourages collaboration at each phase of the application lifecycle—architecture and design, development, testing, implementation and management—and that leverages the same tools and test assets across each phase of the lifecycle.

The HP lifecycle approach to diagnostics has improved our customers' ability to diagnose and resolve performance problems. With our lifecycle approach and software, you can prevent application downtime, leverage tools and skills more effectively across your development, testing and operations teams and optimize the performance of your .NET applications in both pre-production and production environments.

This white paper examines the flaws of traditional approaches to managing, load testing and diagnosing problems in .NET applications and outlines the advantages of combining load testing with lifecycle diagnostics. It also offers practical advice and suggestions on how to transition to the lifecycle diagnostics model.

Checklist:
You should adopt lifecycle diagnostics if you have:

- Difficulties pinpointing the root causes of performance problems
- Unexpected discrepancies between pre-production and production performance
- Limited application visibility for diagnosing and resolving problems
- Poor communication among your architects, developers, QA engineers and operations teams
- Applications experiencing poor or deteriorating performance
- Frequent complaints from end users about sluggish or unavailable applications
- Frequently missed SLAs

Shortcomings of traditional .NET application management and performance testing

Traditional system management products provide visibility into the .NET Common Language Runtime (CLR) by using performance-monitoring application programming interfaces (APIs) to gather CLR-specific metrics, such as .NET CLR exceptions, or by using .NET CLR networking along with system and network metrics. While you need these metrics to understand system performance, they are insufficient for diagnosing problems that can affect your end users. In production, application owners who do not have diagnostics

capabilities often use CLR Debugger (DbgCLR.exe) or Runtime Debugger (coredbg.exe) to diagnose problems. Unfortunately, operations teams usually don't allow them to install a profiler on a production machine. So what should you do if you need to diagnose problems that occur in production? Would load testing these .NET applications earlier in the lifecycle have helped?

Ask ten different companies to describe when and how they load test .NET applications, and you can get ten different answers. Simply put, there is no "traditional" approach, and that is part of the problem. According to industry analysts, approximately 35 percent of companies still use ad-hoc processes for performance management, such as disjointed e-mail messages, spreadsheets and word processors. An additional 35 percent have "silos" of inconsistent practices used by different teams, departments or lines of business, and only 10 percent have implemented consistent lifecycle processes.

Without a proper load testing methodology, applications can be troublesome when you deploy them in production. This section discusses a few problems that can result from poorly defined, piecemeal or inconsistent load testing practices.

Applications pass load tests but fail in production

There can be a large discrepancy between the way an application performs in pre-production load testing and the way it performs in a production environment. Diagnosing the root cause of a performance problem often yields a most unwelcome conclusion: the performance problem occurs early in the design of the application and cannot easily be fixed—if it can be fixed at all. Other problems, such as slow stored procedure calls or memory leaks, result because load testing was not done correctly or at all.

Limited collaboration among architects, developers and testers

In many organizations, virtual walls exist among pre-deployment teams (such as architects and developers), QA teams and the operations group that manages the production rollout. Each group views its role as a series of discrete tasks that are independent from another group's responsibilities. This division often leads to finger-pointing among teams, leaving customers or end users waiting for corrections.

Lack of visibility into applications

Many potential causes can result in performance problems for your .NET applications. Problems typically fall into the following categories:

- Configuration problems
- Code problems
- Application design problems

Configuration problems
Configuration problems include hardware and software configuration at both the system level and the application level. For example, buffering, session timeout, application protection levels and logging configuration can impact your .NET application performance under load.

Code problems
Code problems are typically introduced during application design and development or when updating a bug fix. Code problems can affect the performance of an application under load and often require expensive corrections or verifications. These problems include:

- SQL queries or stored procedure calls that take a long time; for example, nested selects and complex joins
- High-latency synchronous I/O calls; for example, web services calls
- Poor algorithmic performers or resource hogs, such as complex computations
- Excessive logging to disk; for example, log-on exceptions
- Throwing exceptions in a high-volume transaction
- Excessive allocation of collection objects; for example, ArrayList

Application design problems
Application design problems can vary from application to application. Typical problems include:

- Inadvertent copying of data between remote components
- Chatty APIs
- Presentation layers filtering data instead of using effective database queries
- Use of third-party components
- Improper use and storage of data in user sessions

In order to isolate, triage and diagnose these types of problems, you should turn the application black box into a white box and gain visibility inside the application—whether you are doing this in the production, testing or development environment. You need to view the following under load conditions across the lifecycle:

- Layer breakdowns (ASP.NET, ADO and other)
- Method names, latencies, invocation counts, parameter values
- Call chain of methods
- Memory diagnostics
- SQL queries
- Stored procedure calls
- .NET CLR performance counters

In many cases, performance problems only become apparent under load conditions. Unfortunately, developers often have little or no visibility into application performance under load, because QA has jurisdiction, and QA usually does not have the application domain expertise, the software or the data to pass on diagnostic information in a form that's useful to developers.

Moreover, .NET applications are a complex mix of components, often distributed across multiple computer systems. Without visibility into an application under real load conditions, finding and resolving performance issues take longer and cost more than they need to. Similarly, if architects do not understand the software and methods—such as virtual users—that load testers use, they cannot plan the testability of their applications under load adequately. You need to provide greater visibility for your applications and the test process to all your teams at all stages of the application lifecycle.

Lack of integration

Beyond the organizational issues that limit collaboration in performance testing or detecting problems in a production environment, you may have technological challenges. Different teams use different software, processes and infrastructure, creating silos of load testing practices and production monitoring practices. Likewise, each group develops its own skill sets, many of which become specialties that are not accessible to other teams or other lines of business. As a result, you may have missed opportunities for information sharing, redundant infrastructure purchases and inefficient use of software and technologies.

The HP solution: lifecycle diagnostics

Ad-hoc or piecemeal approaches to monitoring, load testing and diagnostics for your .NET applications can lead to inefficiencies, expensive surprises and worse—frustrated end users, low productivity and lost business opportunities. HP has a more efficient alternative: a single unified lifecycle diagnostics approach that works in your production, staging, load testing and development environments.

Overview

The HP lifecycle approach to diagnostics is based on the notion that no aspect of performance testing and tuning .NET applications is a discrete, independent task. You need to view application management, load testing and diagnostics as integral parts of the application lifecycle—elements that require collaboration and information sharing every step of the way, from pre-production application design and development to post-production application management. By providing a diagnostics solution that fully integrates with state-of-the-art monitoring and load testing, you can significantly reduce the mean time to resolution (MTTR) of your application performance problems.

HP Diagnostics software for .NET

HP provides application delivery and application management software for enterprise testing, performance tuning and application monitoring that works with .NET out of the box. The core offering for managing, monitoring, diagnosing and resolving critical problems in .NET applications is HP Diagnostics software for .NET.

HP Diagnostics for .NET is a top-down, end-to-end and business-process-focused, lifecycle approach to .NET diagnostics—in both pre-production and production environments. Your performance testing teams, operations, application support teams and developers can use the software to find application and configuration-level issues in .NET environments, down to method and SQL statement levels, while providing a platform for collaboration that unifies the teams. HP Diagnostics for .NET captures data that can be shared at all phases of the application lifecycle; for example, it provides layer breakdown to help identify hot spots, method-level performance metrics to identify slow methods and a call tree with business context to simplify triage and diagnostics.

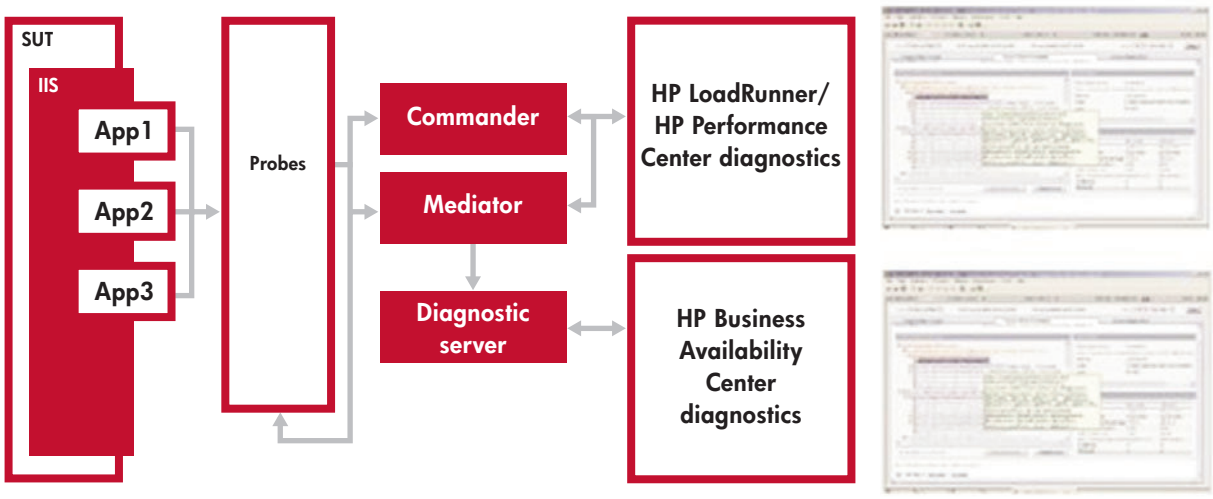


Figure 1. The .NET probe installs on the system under test (SUT) and captures application performance metrics in a CLR-hosted environment. It aggregates and presents the data via the integrated diagnostics user interface in HP LoadRunner, HP Performance Center or HP Business Availability Center.

How HP Diagnostics for .NET works

HP Diagnostics for .NET is built on a diagnostics infrastructure that can work in heterogeneous enterprise environments. A .NET probe uses run-time instrumentation to capture application performance metrics in any CLR-hosted environment, including ASP.NET applications. The software supports .NET v1.1 and IIS 5.1/6.0 on Windows® 2000, XP and Windows 2003 Server. It provides low-overhead diagnostics and works in pre-production and production environments under load conditions.

When the probe initializes, it reads its configuration data and selectively instruments the application at the byte code level at pre-JIT compile time. The probe has built-in instrumentation for out-of-the-box functionality of ASP.NET applications and advanced selective instrumentation capabilities, such as caller-side instrumentation.

With caller-side instrumentation, you can specify some callers of a specific business method while ignoring others. For example, if both the finance application in the namespace **com.mycompany.apps.finance** and the HR application in the namespace **com.mycompany.apps.hr** invoke the business method "findCustomer", you can set the probe to instrument only when the finance application invokes "findCustomer" but not when the HR application invokes it.

This avoids the overhead due to capturing performance metrics on the "findCustomer" method when areas of the application that are not problematic invoke it. You simply specify a "scope" value to include the caller namespace. This capability gives you flexibility and extensibility of instrumentation while lowering overhead costs.

At run time, the probe captures performance data and reports it to the mediator. The mediator aggregates the data, which then displays via the integrated diagnostics user interface in HP LoadRunner software, HP Performance Center software or HP Business Availability Center software.

You can leverage the HP software for other environments, such as J2EE. With the advent of web services and adoption of service-oriented architecture (SOA), cross-technology diagnostics becomes very easy with a heterogeneous solution that can work across differing technology platforms.

How HP Diagnostics for .NET works: a case study

Facilitating collaboration among teams to solve problems can be challenging. With HP Diagnostics for .NET, you can collaborate while isolating problems to avoid finger-pointing among teams. For example, a QA engineer discovers that the performance of a .NET application begins to degrade exponentially at the 500-user level. Without HP Diagnostics for .NET, the QA engineer may not have visibility into why the application's performance is degrading.

With HP Diagnostics for .NET, the QA engineer can triage the problem and determine whether the issue relates to application logic, queuing, disk access or something else. After identifying the type of problem, the QA engineer can drill down deeper, for example, to determine that the problem is in the ADO layer. Now the QA engineer knows which developers to talk to. The developers can see the data from the drill-down and determine which method is the slowest, how long it's taking and when and where it's being called.

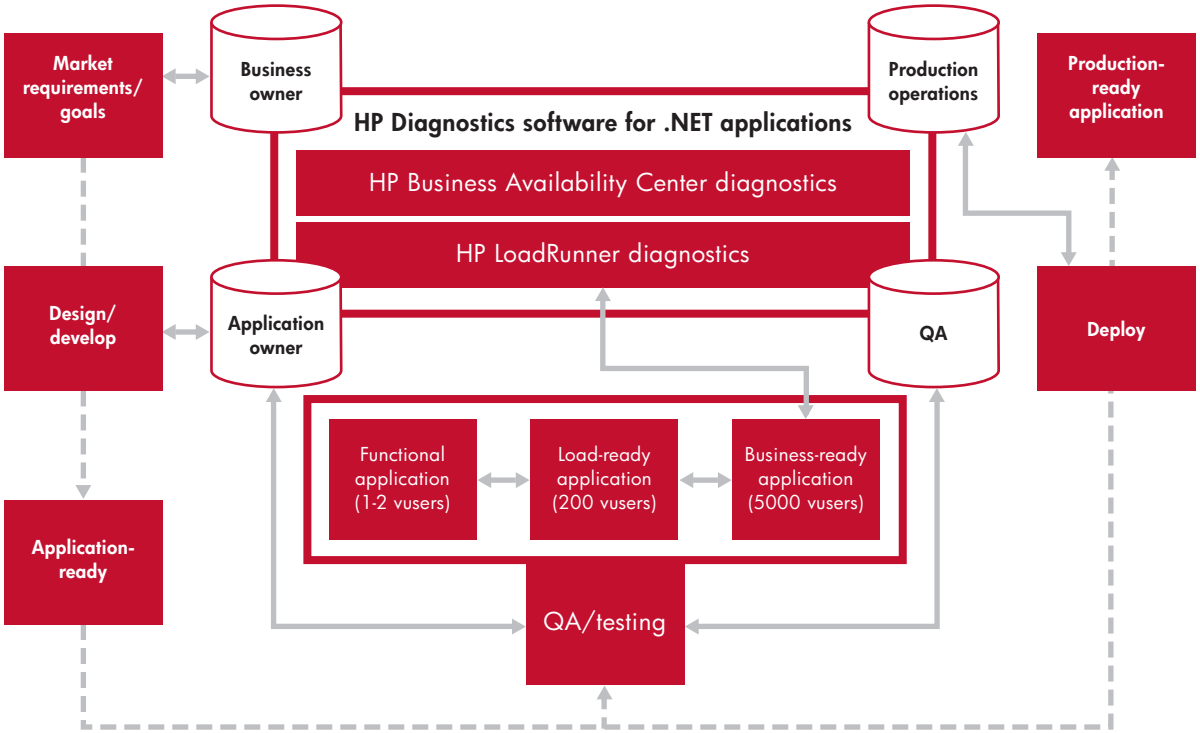


Figure 2. HP Diagnostics for .NET supports collaboration across your organization in diagnosing and troubleshooting application performance problems.

When a problem occurs in a production environment, you need the same type of collaboration. For example, a level 2 application support engineer can share data with a developer for a problem that occurred at 2 a.m. on Saturday.

Let's look at an example using HP Diagnostics for .NET. Application performance problems affect four roles: the business owner, the application owner, QA and production operations.

The business owner drafts market requirements and business goals and provides them to the development team to develop an application. Once the application is designed and developed, the QA and testing phase commences. HP Diagnostics for .NET facilitates effective collaboration using HTML reports and offline analysis data so that the application meets business requirements.

This application can now be launched into production, and the production team monitors and diagnoses problems, using the same diagnostics components deployed in pre-production. Thus, the integrated lifecycle diagnostics facilitates collaboration, reduces down time due to finger-pointing and reduces the MTTR of problems—no matter when they occur.

This process works only if the development team is committed to fixing problems as QA and Operations discover them and if project schedules include this time. HP diagnostic capabilities, along with industry-standard load testing and management software, enhance collaboration across the enterprise.

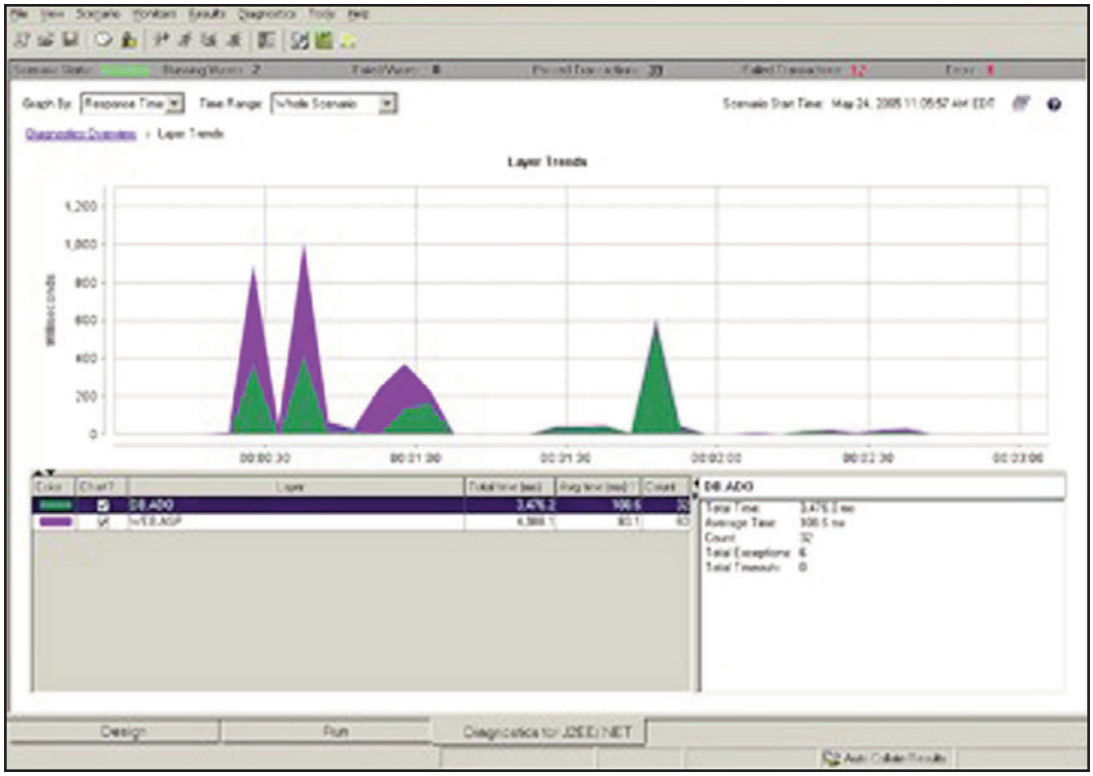
Methodology and best practices

Using the right method helps your organization manage chaos and function better as a team. When managing and testing your .NET applications, follow these guidelines for better results.

Start with the business process

Testing with a focus on the business process lets you deliver on and measure the results against your business goals. When load testing your application, you should create load scripts that include critical business processes, defined by synthetic transactions. For example, you can create a script for when a user logs in, browses for products, adds items to a cart, makes a purchase and then logs out. You can organize all of these end-user actions into one business process and scale it to several thousands of users. Load testing with the business process in mind lets you diagnose application problems in the context of the business process that is performing poorly and helps you prioritize the diagnostics tasks.

Figure 3. The Layer Breakdown view gives you detailed information about performance problems by application layer, including total exceptions.



Track real users

While using synthetic users is excellent for proactive monitoring and diagnostics, you also need to capture and collect data from real users of your application. HP Diagnostics for .NET supports both synthetic and real-user transaction breakdowns to get diagnostic information at the method and SQL level.

Model loads appropriately

Testing an application with the wrong load characteristics can be problematic or render your tests worthless. Make sure you understand the business needs for scaling the application by working with the business and application owners. Get a good understanding of sustained peak loads and load bursts for your business.

For example, if you are an insurance company, open enrollment periods for your customers can cause significant load on your systems during specific times of the year and times of day. Your load scenarios should reflect trends in loads. Industry-standard load testing and application management software from HP let you model complex loads and user behavior while giving you the flexibility of modeling different client bandwidths, cache settings and others, making your load tests more meaningful.

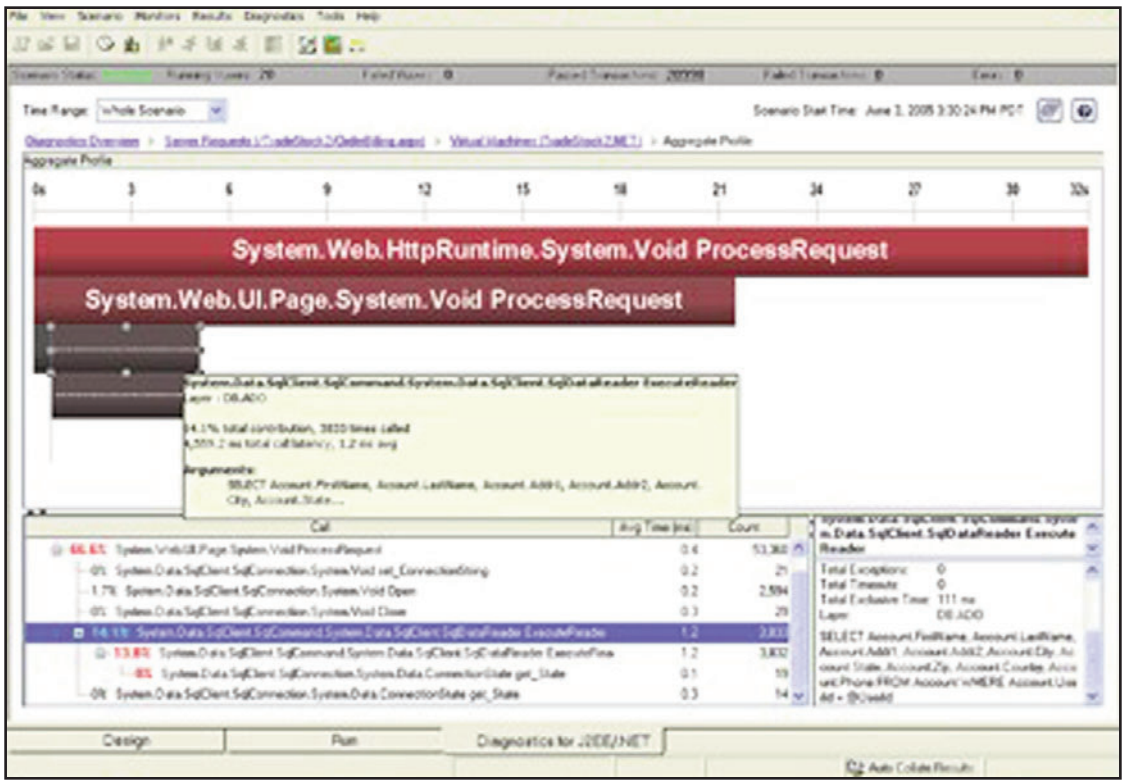
Think in layers

When load testing your .NET applications, collaborate with your development teams to identify the "weak links" in the application from a performance perspective. These areas can include caching mechanisms, persistence layers and third-party service calls made by the application. When load testing your applications, capture diagnostics information for these layers so that you can triage the application problems more easily. HP Diagnostics for .NET provides ASP.NET and ADO.NET as default out-of-the-box layers. In addition, you can easily extend the software to include your application-specific layers in the probe configuration. Once you identify these layers, make them part of your QA process and your production application management framework by migrating the lifecycle probe across your application lifecycle.

Triage and diagnose

Triage performance problems by first eliminating any systemic problems, such as hardware or operating system configuration. For example, a rogue batch process can take most of the CPU of a system, severely impacting your application performance. Once you eliminate other possibilities using HP LoadRunner monitors and analysis, you can use HP Diagnostics for .NET to capture diagnostics information.

Figure 4. HP LoadRunner lets you see how business transactions perform under load and determine where the application time is spent for selected business processes.



During the diagnostics phase, identify which business processes are slow. If your application operates in a clustered environment, drill down to isolate which specific VM is slow, and then generate a call tree that shows you where time was spent for the selected business process and VM. Identify the slow methods, SQL queries or other problem areas and bring in other team members (such as developers) as necessary to collaborate and correct the problems.

Adopt a lifecycle approach

A lifecycle approach is key to reducing the MTTR of problems. It allows you to resolve problems quicker and focus your resources and time on key initiatives instead of firefighting on a daily basis. Using the same diagnostics technology across the application lifecycle provides several key advantages:

- Better collaboration because different teams look at the same data provided by a single probe
- Learn once, and you can use the knowledge across the lifecycle
- Easier maintenance and deployment

For example, when you create your business process scripts to test the performance of your application, you can use the same scripts in production to manage the application—and vice versa.

Use the Layer Breakdown view to triage the issues to a specific application layer. The details section on the first screen shows the exceptions for an application as well.

Once you determine that a layer is slow, capture the screen shot of the report and include it in a bug report. You can also view how the business transactions perform under load and see where time is spent inside the application for a selected business process.

If you run tests overnight or longer, the diagnostics information for your application is available in the analysis module of HP LoadRunner. You can generate Word documents from the various HP LoadRunner reports, present them to your development teams, and maintain the document as a document of record for the tests you run and the application problems you find. As a result, your development teams can collaborate with your QA teams to view detailed data, such as memory and method level details, in order to diagnose application problems.

Plan for application management and performance testing

Applications often suffer from severe performance problems because performance testing is not part of the project plan—or, in some cases, is an afterthought. In most reactive situations, the problem is too expensive to find, fix and validate. You should include time in your project plan to account for performance testing prior to going live.

At the business level, you can include performance testing as part of your "go-live" planning. At the QA level, you can include performance testing as part of your "build certification" planning. At the development level, you can include performance testing as part of your "release-to-QA" planning. If you are working with third-party vendors to develop your application, performance testing can be part of the project deliverable. Work with your application owners and development teams to take responsibility for addressing any application-specific issues discovered during the process.

Real-world example

As more .NET applications go live into production and as technology complexity increases, there is also an increase in the number of points of failure that a business process can encounter in the enterprise. Diagnosing application problems with a business process context becomes vital in helping you prioritize the problems. The following example shows how HP Diagnostics for .NET supports collaboration to triage and diagnose a difficult problem.

A large bank was completing a mission-critical .NET application for production deployment. The business goal for this application was to scale to 2,500 users with a given set of hardware. All went well throughout the application lifecycle until load testing the application. Using HP Diagnostics for .NET with HP LoadRunner, the bank created a test case that focused on the business processes and modeled the load for the business need. By the end of the load test, the bank determined that the application could not scale to 500 users, despite using twice the amount of hardware originally allocated—a cause for serious concern.

QA engineers quickly enabled HP Diagnostics for .NET within HP LoadRunner and re-ran the load test. They found several problems that were missed throughout the application lifecycle. They learned that performance problems often occur under load conditions and that their project plans should include time for performance testing as part of their go-live plans.

When QA engineers triaged the problem, they noticed that the "Generate Report" business process was intermittently slow. Using the offline analysis and HTML reports of HP LoadRunner, QA collaborated with the development team to determine where time was spent inside the application while under load for the business process.

HP Diagnostics for .NET showed several issues from inside the HP LoadRunner controller:

- The application was not following the anticipated path of execution.
- An application-specific method was invoking a third-party DLL to create a PDF file, slowing down the business transaction significantly.
- Bad SQL queries contributed to the sluggish performance of the application under load.

All of the above caused the application to have poor performance under load. Integrating HP LoadRunner with HP Diagnostics for .NET helped the QA engineers quickly isolate and triage problems while collecting necessary diagnostic data for the development teams. Developers had the appropriate data to determine the root cause of the problems and correct them quickly. As they made corrections, HP LoadRunner helped them verify and regress the corrections by simply re-running the load tests with the correct business process.

As shown in this real-world example, detecting problems earlier in the lifecycle can reduce stress during the go-live process and can help you to deploy an application that meets your business needs.

HP Diagnostics for .NET is powerful software for managing, load testing and diagnosing different types of performance issues. It also provides your QA engineers with a way to collaborate and supply data to your developers to help them resolve performance problems quickly. Simply put, it's a means of getting the right information, in the right form, to the right people.

HP Diagnostics for .NET features and benefits

- **Unified lifecycle probe.** You can use the same .NET probe across the lifecycle to diagnose performance issues discovered during pre-production testing and production operations. A single data-capturing mechanism spanning the lifecycle reduces the number of products you need and encourages collaboration throughout the process.

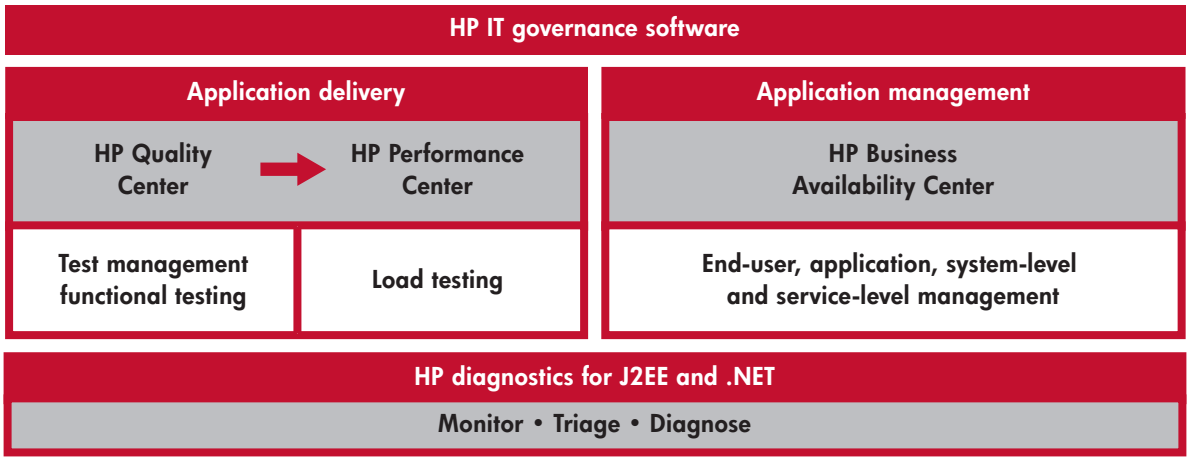


Figure 5. HP BTO software addresses the application lifecycle for your .NET applications and lets you monitor, triage and diagnose problems at every phase, helping you identify and resolve performance issues faster.

- **Diagnostics triage.** HP Diagnostics for .NET starts from the end-user business process, breaking down the time spent among back-end components, application components and methods to pinpoint the root cause of problems. This helps you prioritize and focus on the problems with the highest business impact. It reduces your MTTR through intuitive, guided diagnostics.
- **In-depth application-level visibility.** HP Diagnostics for .NET shows the response-time breakdown by .NET tier and method without source-code modification, offering full visibility inside .NET applications.
- **360-degree monitoring.** Monitoring from end-user, application and system perspectives lets you proactively detect issues in the application and in all connected systems and environments.
- **Low overhead.** Unlike development profilers, HP Diagnostics for .NET is designed for real-world user load conditions with low overhead.
- **Support for clustered environments.** Multi-CLR support enables intuitive visualization and easy comparison of performance across a cluster of application server instances, allowing for rapid identification of problems related to load-balancing.
- **Leverage HP LoadRunner investments.** Customers who use HP LoadRunner to optimize performance can leverage the investment to add diagnostics into their QA processes. Using a common toolset and business scripts, development and QA teams can more quickly identify performance problems proactively, diagnose their root causes, resolve them in less time and prevent many performance problems from happening before deployment.

Integration with other HP BTO software

HP Diagnostics for .NET works with other HP business technology optimization (BTO) software, including HP Quality Center, HP Performance Center and HP Business Availability Center, to assist customers in optimizing the entire performance lifecycle process. You can implement these internally or deliver them through HP Services.

HP Quality Center is web-based software for performing quality assurance across a wide range of IT and application environments. It includes integrated, role-based software and best practices as well as an open, scalable and extensible foundation. HP Quality Center can optimize and automate key quality activities, including requirements, test and defects management, functional testing and business process testing.

HP Performance Center is an integrated set of software that automates performance optimization processes, such as bottleneck identification and diagnostics. It includes HP LoadRunner, which generates consistent, measurable and repeatable load tests from a single point of control, and HP diagnostics software for resolving performance issues in J2EE, Oracle, SAP and Siebel applications as well as .NET applications.

HP Business Availability Center includes a unified dashboard for managing IT operations to meet business objectives. It lets you view key business process and system indicators in real time from an end-user, business-level and service-level perspective, and lets you take a "center-of-excellence" approach for enabling your production applications to meet your service-level goals and deliver business results.

Incorporating lifecycle diagnostics into existing processes

At most companies today, existing QA processes are bound by two things: a lack of application domain expertise within QA, because development and testing have been viewed as discrete, independent functions; and lack of time, because many projects are both urgent and more time-consuming than their planners expected.

The consequence is that too often, application projects are "thrown" into production with insufficient load testing. The lifecycle diagnostics approach, along with next-generation testing tools, can help address both of these issues. But the question remains: What is the right way to implement the lifecycle model into existing QA processes? This section offers specific suggestions.

1. Involve your QA team early in application design.
When you involve QA and performance teams early in the application design, they are better equipped to create test cases that stress a weak area in the design from a performance perspective. For example, a caching tier that is untested for performance can create problems if the cache size has become unmanageable or the cache becomes too slow.

2. Include additional time in your schedule for performance and load testing your applications.
Work with architects and business owners to establish application performance objectives. See that load testing addresses these performance objectives and that the development project plan includes correcting found bugs before releasing the application into production.

3. Encourage your architects to invite QA early into the design and development phase.
Understand from your architects whether the application is sensitive to multiple users or threads accessing a certain functional area (for example, the query engine to the database). In such cases, performance engineers can use the rendezvous points in HP LoadRunner to have many simultaneous users stress a specific application area.

4. Designate someone to own and drive the lifecycle diagnostics initiative.
The first step is to identify who owns the end users' experience. Typically, the owner is the part of your organization that is most affected when applications are unavailable or perform poorly. This group can benefit the most from the move to lifecycle diagnostics. The initial push needs to come from a senior operations executive within IT or a specific line of business, supported by application teams. Executive buy-in is critical at this phase. Your lifecycle diagnostics initiative depends on the champion's ability to prove the value of this approach over the limitations and challenges of traditional approaches.

Conclusion

Increased collaboration at all phases of the application lifecycle can help identify and resolve many common causes of performance problems in .NET applications. Many organizations believe that the "walls" among developers, QA engineers and operations teams are a political reality, attributable to human nature. Perhaps, but it is also quite possible that armed with better technology, better software and a lifecycle approach to diagnostics, these teams can achieve a higher level of communication and collaboration—and a higher level of application performance and end-user satisfaction.

To learn more, visit www.hp.com/go/software

4AA1-4102ENW, August 2007

invent