TP322, 06/02

Rational. software

IBM

End-to-End Testing of IT Architecture and Applications

Jeffrey Bocarsly
Division Manager, Automated Functional Testing
RTTS

Johanthan Harris
Division Manager, Scalability Testing
RTTS

Bill Hayduk
Director, Professional Services
RTTS

Table of Contents

Introduction

Not long ago, industry-standard testing practices, which had evolved in response to quality issues facing the client/server architecture, centered either on the client for front-end functional tests, or the server for back-end scalability and performance tests. This "division of labor" derived largely from the fact that the classic client/server architecture, a two-tier structure, is relatively simple compared to current multi-tier and distributed environments. In the standard client/server arrangement, issues are either on the client side or on the database side.

Today, however, the typical computing environment is a complex, heterogeneous mix of legacy, homegrown, third party, and standardized components and code (see Figure 1). Since the advent of the Web, architectures have increased in complexity, often with a content tier placed between one or more back-end databases and the user-oriented presentation tier. The content tier might deliver content from multiple services that are brought together in the presentation tier, and might also contain business logic that previously would have been found in the front end of a client/server system.

This increase in complexity, overlaid with the problems of integrating legacy and cutting-edge development, can make the characterization, analysis, and localization of software and system issues (including functional and scalability/performance problems) major challenges in the development and delivery of software systems. Further, with the acceptance of SOAP/XML as a standard data transmission format, issues of XML data content have become increasingly crucial on both .NET and J2EE development platforms. Simply put, the complexity of current architectures and computing environments has rendered the original client/server-oriented testing scheme obsolete.

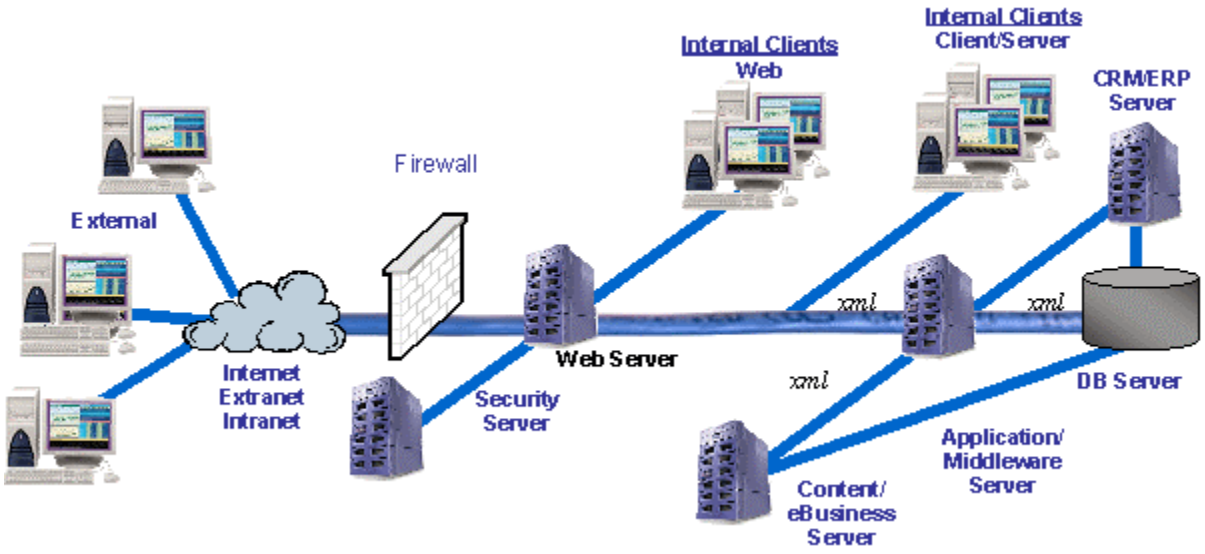

Figure 1: A Typical Multi-Tier Architecture Today

An Overall Quality Strategy

Clearly, new, aggressive quality enhancement strategies are necessary for successful software development and deployment. The most potent strategy combines testing the environment's individual components with testing the environment as a whole. In this strategy, testing at both the component and system levels must include functional tests to validate data integrity as well as scalability/performance tests to ensure acceptable response times under various system loads.

For assessment of performance and scalability, these parallel modes of analysis aid in determining the strengths and weaknesses of the architecture, and in pinpointing which components must be involved in resolving the performance- and scalability-related issues of the architecture. The analogous functional testing strategy, full data integrity validation, is becoming increasingly critical, because data may now derive from diverse sources. By assessing data integrity -- including any functional transformations of data that occur during processing -- both within components and across component boundaries, testers can localize each potential defect, making the tasks of system integration and defect isolation part of the standard development process. *End-to-End Architecture Testing* refers to the concept of testing at all points of access in a computing environment and combines functionality and performance testing at the component and system levels (see Figure 2).

In some ways, End-to-End Architecture Testing is essentially a "gray box" approach to testing -- a combination of the strengths of white box and black box testing. In white box testing, a tester has access to, and knowledge of, the underlying system components. Although white box testing can provide very detailed and valuable results, it falls short in detecting many integration and system performance issues. In contrast, black box testing assumes little or no knowledge of the internal workings of the system, but instead focuses on the end-user experience -- ensuring the user is getting the right results in a timely manner. Black box tests cannot typically pinpoint the cause of problems. Nor can they ensure that any particular piece of code has been executed, runs efficiently, and does not contain memory leaks or other similar problems. By merging white and black box testing techniques, End-To-End Architecture Testing eliminates the weaknesses inherent in each, while capitalizing on their respective advantages.

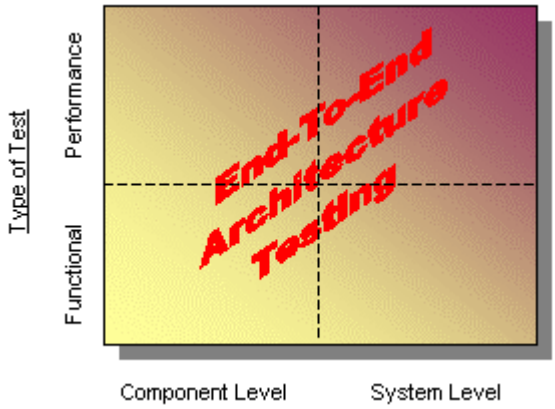

Figure 2: End-to-End Architecture Testing Includes Functional and Performance Testing at All Points of Access

For performance and scalability testing, points of access include hardware, operating systems, applications databases, and the network. For functional testing, points of access include the front-end client, middle tier, content sources, and back-end databases. With this in mind, the term *architecture* defines how all of the components in the environment interact with other components in the environment, and how users interact with all of these components. Individual components' strengths and weaknesses are defined by the specific architecture that organizes them. It is the uncertainty of how an architecture will respond to the demands placed on it that creates the need for End-to-End Architecture Testing.

To implement End-to-End Architecture Testing effectively, RTTS has developed a successful, risk-based test automation methodology. The Test Automation Process (TAP) is based upon years of successful test implementations, utilizing best-of-breed automated test tools. It is an iterative approach to testing with five distinct phases:

- Project assessment
- Test plan creation or improvement
- Test case development
- Test automation, execution, and tracking
- Test results evaluation

The individual functional and performance tests required for End-to-End Architecture Testing are conducted in the "test automation, execution, and tracking" phase. As shown in Figure 3, this phase is repeated, and the associated tests are refined, with each iteration of the process.

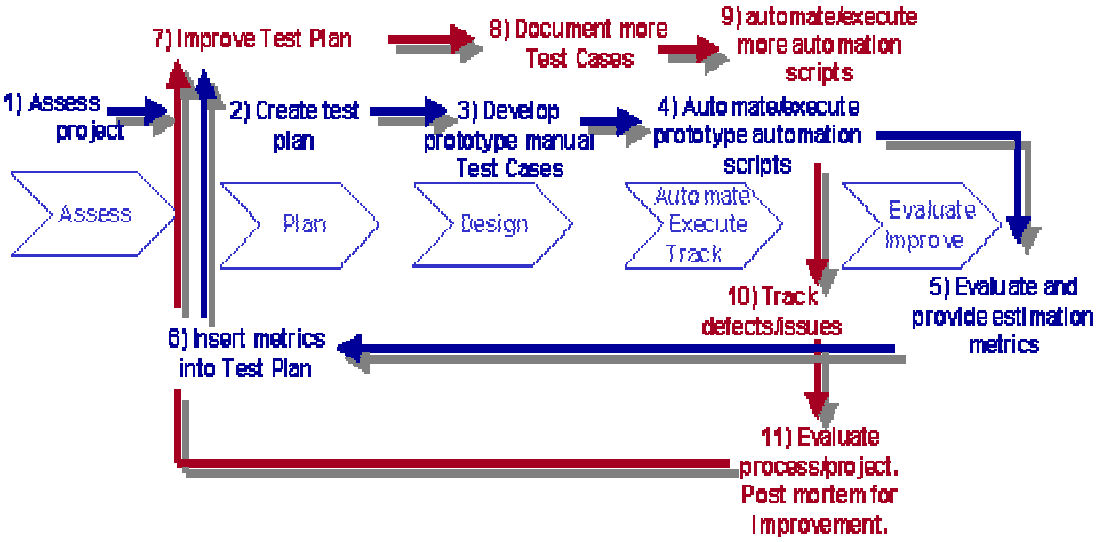

Figure 3: The RTTS Test Automation Process (TAP) for End-To-End Architecture Testing

Component Level Testing

Clearly, individual components must be developed before they can be assembled into a functioning system. Because components are available for testing early on, End-to-End Architecture Testing in TAP begins with *component testing*. In component testing, appropriate tests are conducted against individual components as the environment is being built. Both functional and scalability testing at the component level are exceptionally valuable as diagnostics to help identify weak links before and during the assembly of the overall environment.

Functional Tests at the Component Level

Functional testing applied at this level validates the transactions that each component performs. This includes any data transformations the component or system is required to perform, as well as validations of business logic that apply to any transaction handled by the component. As application functionality is developed, *infrastructure testing* verifies and quantifies the flow of data through the environment's infrastructure, and as such, simultaneously tests functionality and performance. Data integrity must be verified as data begins to be passed between system components. For example, *XML Testing* validates XML data content on a transaction-by-transaction basis, and where desirable, validates formal XML structure (metadata structure). For component tests, an automated and extensible testing tool like IBM Rational® Robot can greatly reduce the amount of time and effort needed to drive GUI tests as well as functional tests on GUI-less components. Rational Robot's scripting language offers support for calling into external COM (Component Object Model) DLLs, making it an ideal tool for testing GUI-less objects either directly or via a COM test harness. Also, the new Web and Java testing functionality in Rational Suite® TestStudio® and Rational® TeamTest provides additional capabilities for testing J2EE architectures and writing or recording test scripts in Java.

Scalability and Performance Tests at the Component Level

In parallel to these functional tests, scalability testing at this level exercises each component within the environment to determine its transaction (or volume) limitations. Once enough application functionality exists to create business related transactions, *transaction characterization testing* is used to determine the footprint of business transactions -- including how much network bandwidth is consumed, as well as the CPU and memory utilization on back-end systems. *Resource testing* expands on this concept with multi-user tests conducted to determine the total resource usage of applications and subsystems or modules. Finally, *configuration testing* can identify what changes in hardware, operating system, software, network, database, or other configurations are needed to achieve optimal performance. As with functional testing, effective automated tools such as those found in Rational Suite TestStudio and Rational TeamTest can greatly simplify scalability and performance testing. In this case, the ability to create, schedule, and drive multi-user tests and monitor resource utilization for resource, transaction characterization, and configuration testing is essential to efficiently and successfully complete these tests.

System Level Testing

When the system has been fully assembled, testing of the environment as a whole can begin. Again, End-to-End Architecture Testing requires verification of both the functionality and performance/scalability of the entire environment.

Functional Tests at the System Level

One of the first issues that must be considered is that of integration. *Integration testing* addresses the broad issue of whether the system is integrated from a data perspective. That is, are the hardware and software components that should be talking with one another communicating properly? If so, is the data being transmitted between them correct? If possible, data may be accessed and verified at intermediary stages of transmission between system components. These points may occur, for example, when data is written to temporary database tables, or when data is accessible in message queues prior to being processed by target components. Access to data at these component boundaries can provide an important additional dimension to data integrity validation and characterization of data issues. For cases in which data corruption can be isolated between two data transmission points, the defective component is localized between those points.

Scalability and Performance Tests at the System Level

For every question that can be asked about how an environment scales or performs, a test can be created to answer that question.

- How many users can access the system simultaneously before it can no longer maintain acceptable response times?
- Will my high-availability architecture work as designed?
- What will happen if we add a new application or update the one I currently use?
- How should the environment be configured to support the number of users we expect at launch? In six months? In one year?
- We only have partial functionality -- is the design sound?

Answers to these questions are obtained through a wide range of testing techniques, including: scalability/load testing, performance testing, configuration testing, concurrency testing, stress and volume testing, reliability testing, and failover testing, among others.

In the area of system capacity, whole environment testing typically begins with *scalability/load testing*. This kind of test places an increasing load on the target environment, until either performance requirements such as maximum response times are exceeded or a particular resource is saturated. These tests are designed to determine the upper limits of transaction and user volume and are usually combined with other test types to optimize performance.

Related to scalability/load testing, *performance testing* is used to determine whether the environment meets requirements at set loads and mixes of transactions by testing specific business scenarios (see Figure 4).

Paralleling configuration testing at the component level, *configuration testing* at the system level provides tradeoff information on specific hardware or software settings as well as metrics and other information needed to effectively allocate resources.

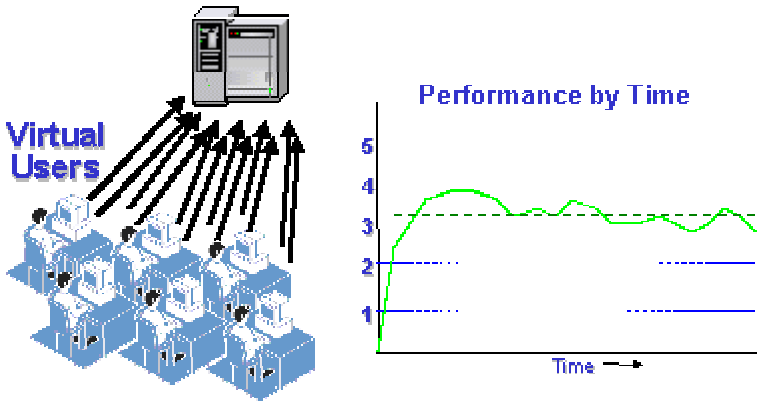

Figure 4: Performance Testing: Will the System Perform as Required with a Specific User Load?

Concurrency testing (Figure 5) profiles the effects of multiple users simultaneously accessing the same application code, module, or database records. It identifies and measures the levels of locking and deadlocking, and use of single-threaded code and locking semaphores in a system. Technically, concurrency testing could be categorized as

a kind of functional testing. However, it is often grouped with scalability/load tests because it requires multiple users or virtual users to drive the system.

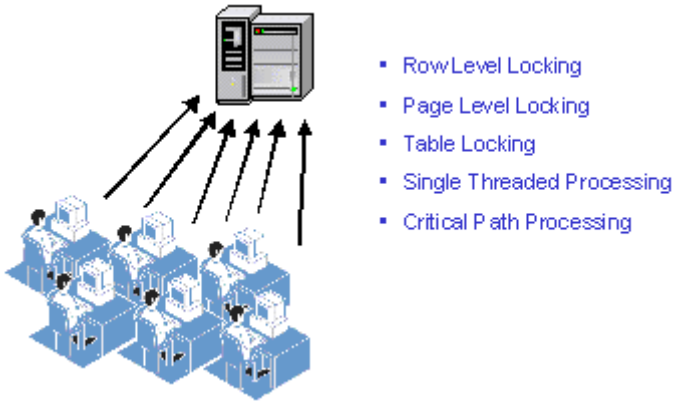

Figure 5: Concurrency Testing Identifies Deadlocking and Other Concurrent Access Problems

Stress testing (Figure 6) exercises the target system or environment at the point of saturation (depletion of a resource such as CPU, memory, etc.) to determine if the behavior changes and possibly becomes detrimental to the system, application, or data. *Volume testing* is related to stress testing and scalability/load testing, and is conducted to determine the volume of transactions that a complete system can process. Stress and volume testing are performed to test the resiliency of an environment to withstand burst or sustained high-volume activity, respectively -- without failing due to defects such as memory leaks or queue overruns.

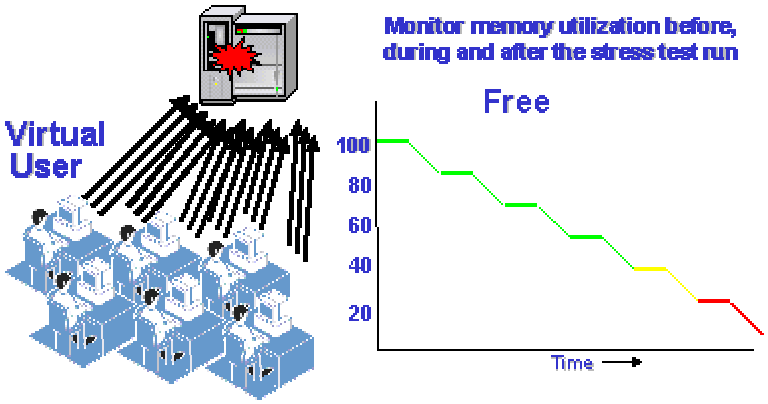

Figure 6: Stress Testing Determines the Effect of High-Volume Usage

Once the environment or application is working and optimized for performance, a long-duration *reliability test* exercises an environment at sustained 75 percent to 90 percent utilization to discover any issues associated with running the environment for extended periods of time.

In environments that employ redundancy and load balancing, *failover testing* (Figure 7) analyzes the theoretical failover procedure, and tests and measures the overall failover process and its effects on the end-user. Essentially, failover testing answers the question, "Will users be able to continue accessing and processing with minimal interruption if a given component fails?"

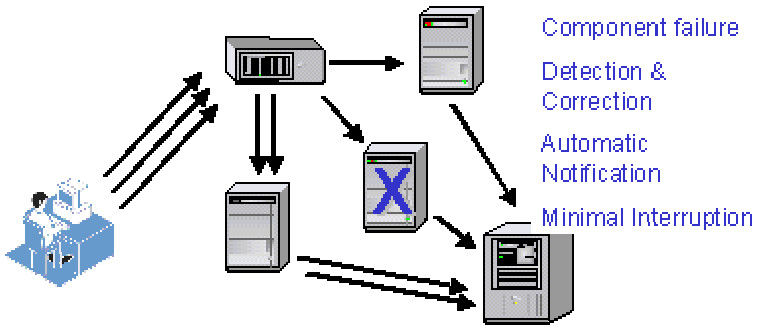

Figure 7: Failover Testing: What Will Hhappen If Component X Fails?

And finally, if an environment employs third-party software or accepts feeds from outside sources or hosted vendors, then SLA (Service Level Agreement) testing can be conducted to ensure end-user response times and inbound and outbound data streams are within contract specifications. A typical agreement guarantees a specified volume of activity over a predetermined time period with a specified maximum response time.

Once external data or software sources are in place, monitoring of these sources on an ongoing basis is advisable, so that corrective action can be taken quickly if problems develop, minimizing the effect on end users.

As with scalability testing at the component level, Rational Suite TestStudio, Rational TeamTest, and similar tools offer advanced, multi-user testing capabilities and can be used to effectively drive many if not all of the above scalability and performance tests.

A Real-World Example

Perhaps the best way of illustrating End-to-End Architecture Testing is through an example. Consider the following scenario:

An eRetailer has built a public Web bookstore that uses four content-providing Web services in its content tier. One of the services provides the catalog, including book titles, blurbs, and authors. A second service provides the current inventory for all products. The third service is the price server, which provides pricing, shipping, and tax information, based on the purchaser's locale, and executes transactions. The final service holds user profiles and purchasing history.

The presentation tier transforms user requests entered through the UI into XML and submits requests to the proper content server. Response XMLs are then transformed to HTML by the presentation layer and served to the user's session. Each of the content-tier services updates the others as needed. (See Figure 8.) For example, the price service must update the profile service as the user's purchasing history changes.

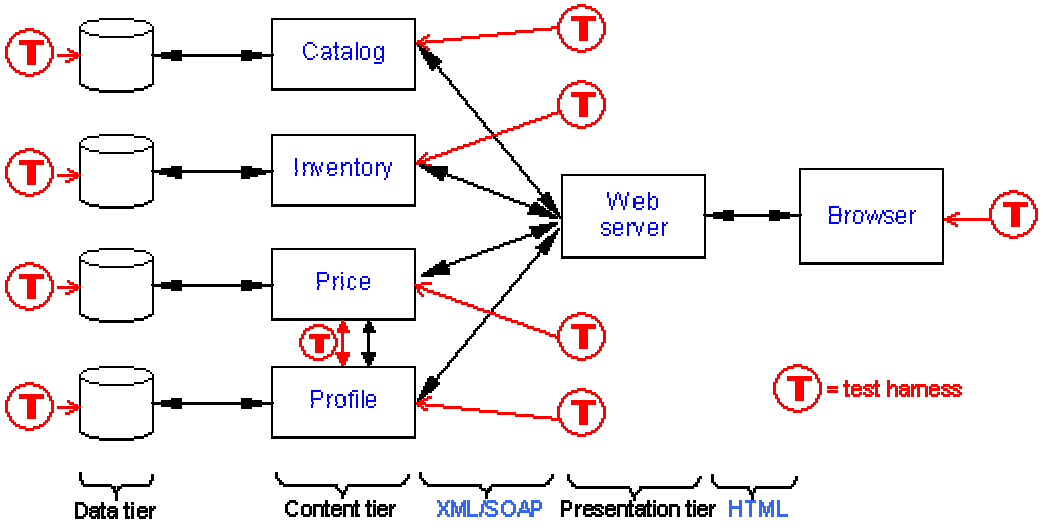

Figure 8: Points of Access for a Typical eRetailer Application

An End-to-End Architecture Testing strategy for the system outlined above starts by applying both functional and scalability/load testing to each of the content-tier systems separately. XML requests are submitted to each of the content services, and the response XML documents are captured and evaluated for either data content or response time. As each of these services is integrated into the system, both functional and scalability/load testing are performed on the assembled system, by submitting transactions to the Web server. Transactions can be validated through the entire site infrastructure, both for functional testing (using SQL queries) and scalability/load testing.

As the system is developed, individual test harnesses, applied at all points of access, can be used to tune each service to function within the whole assembly, both in terms of data-content (i.e., functionality) and performance (i.e., scalability). When issues are discovered in the front end (i.e., via the browser), the test suites and test harnesses that were used to test individual components facilitate rapid pinpointing of the defect's location.

The Benefits of Network Modeling

When it is included as part of the design process -- either prior to the acquisition of hardware or during the initial test phase -- modeling different network architectures can amplify the benefits of End-to-End Architecture Testing by helping make network designs more efficient and less error-prone. Prior to deployment, network infrastructure modeling can help pinpoint performance bottlenecks, errors in routing tables and configurations. In addition, application transaction characterizations obtained during testing can be input into the model to identify and isolate application "chattiness"[1] and potential issues within the infrastructure.

Conclusion

End-to-End Architecture Testing exercises and analyzes computing environments from a broad-based quality perspective. The scalability and functionality of every component is tested individually and collectively during development as well as in prerelease quality evaluation. This provides both diagnostic information that enhances development efficiency and a high degree of quality assurance upon release. End-to-End Architecture Testing provides a comprehensive, reliable solution for managing the complexity of today's architectures and distributed computing environments.

Of course, given the broad range of tests and analysis required, an end-to-end testing effort requires considerable expertise and experience to organize, manage, and implement. But from a business perspective, organizations that embrace an end-to-end testing philosophy will be able to guarantee higher levels of application and system performance and reliability. And ultimately, these organizations will reap the benefits of increased quality: better customer relationships, lower operating costs, and greater revenue growth.

For the past six years RTTS, an IBM Rational Partner, has developed and perfected its End-to-End Architecture Testing approach, working with hundreds of clients to ensure application functionality, reliability, scalability, and network performance. Visit the RTTS Web site at www.rttsweb.com

[1] An application is "chatty" if it requires numerous queries and responses to complete a transaction between components

IBM software integrated solutions
IBM Rational supports a wealth of other offerings from IBM software. IBM software solutions can give you the power to achieve your priority business and IT goals.

- *DB2® software helps you leverage information with solutions for data enablement, data management, and data distribution.*
- *Lotus® software helps your staff be productive with solutions for authoring, managing, communicating, and sharing knowledge.*
- *Tivoli® software helps you manage the technology that runs your e-business infrastructure.*
- *WebSphere® software helps you extend your existing business-critical processes to the Web.*
- *Rational® software helps you improve your software development capability with tools, services, and best practices.*

Rational software from IBM
Rational software from IBM helps organizations create business value by improving their software development capability. The Rational software development platform integrates software engineering best practices, tools, and services. With it, organizations thrive in an on demand world by being more responsive, resilient, and focused. Rational's standards-based, cross-platform solution helps software development teams create and extend business applications, embedded systems and software products. Ninety-eight of the Fortune 100 rely on Rational tools to build better software, faster. Additional information is available at www.rational.com and www.therationaledge.com, the monthly e-zine for the Rational community.

IBM Corporation
Software Group
Route 100
Somers, NY 10589
U.S.A.

Printed in the United States of America
01-03 All Rights Reserved.
Made in the U.S.A.

The IBM home page on the Internet can be found at **ibm.com**