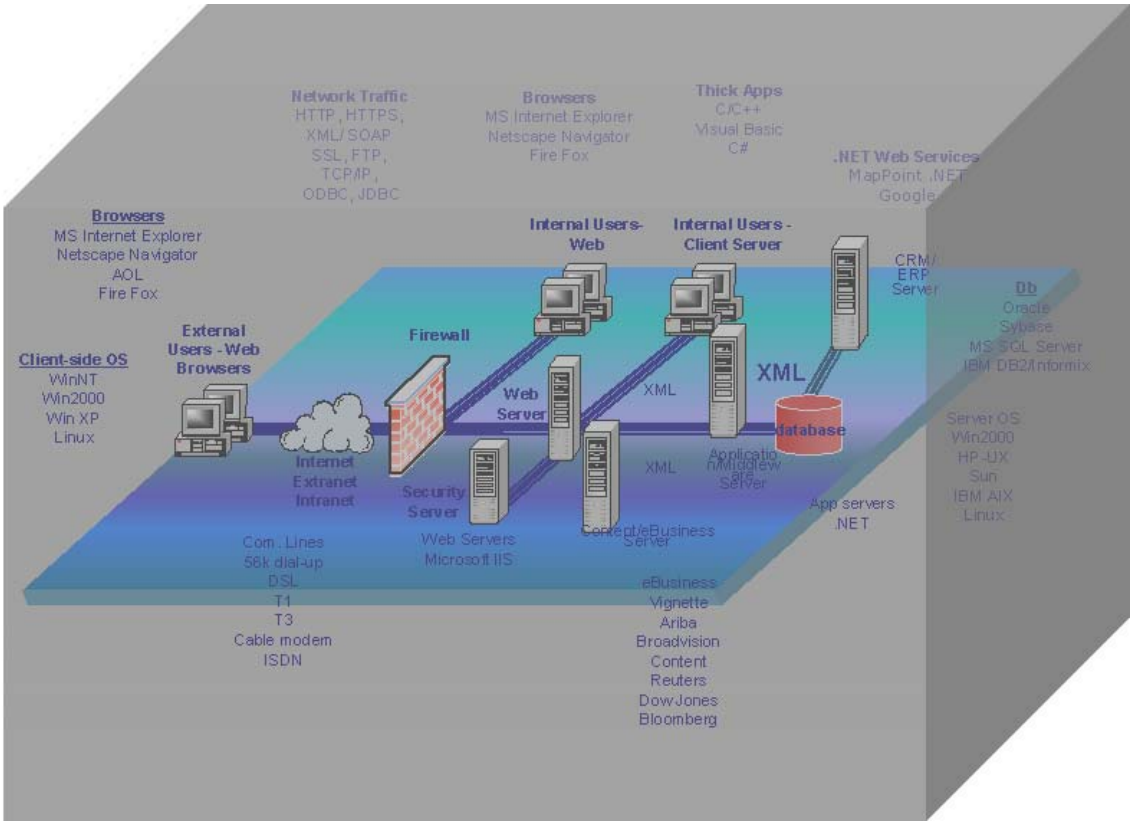
Thinking *Inside* the Box: Greybox Testing of IT Architecture and Applications



Jeffrey Bocarsly, PhD.
Division Manager
Automated Functional Testing
RTTS

Jonathan Harris
Division Manager
Scalability Testing
RTTS

Bill Hayduk
Director
Professional Services
RTTS

Introduction

Greybox architecture testing refers to a mixed approach to software testing that includes key elements of "whitebox" and of "blackbox" testing. Blackbox testing views the entire architecture as a unitary whole, and is *un*interested in the inner workings of the system. Whitebox testing peeks under the hood, and seeks to examine the system at a granular level, often down to the level of code analysis. Greybox architecture testing takes its cues from the whitebox and blackbox approaches, probing architecture under the hood down to the component level, and building up to test the environment at the fully-assembled blackbox level. Greybox testing therefore tests at all points-of-access in a computing environment.

The Trend Toward Complexity

Typical heterogeneous enterprise computing environments today are a complex mix of legacy, custom built, third party and standardized components and code. With the advent of the Web, architectures have increased in complexity, often with a content-tier placed between back-end database(s) and the user-oriented presentation-tier. Service Oriented Architectural (SOA) concepts have provided a framework for additional architectural elaboration. The content-tier (delivered as SOA web services) may deliver content from multiple feeds or services that are brought together in the presentation-tier, and may also contain business logic that previously would have been found in the front-end of a client-server system. Legacy applications may be wrapped as SOA web services, and newer development may as well be exposed to the environment under the SOA paradigm. External applications may pull data from the architecture via message queues, and data may come into the system from automated feeds.

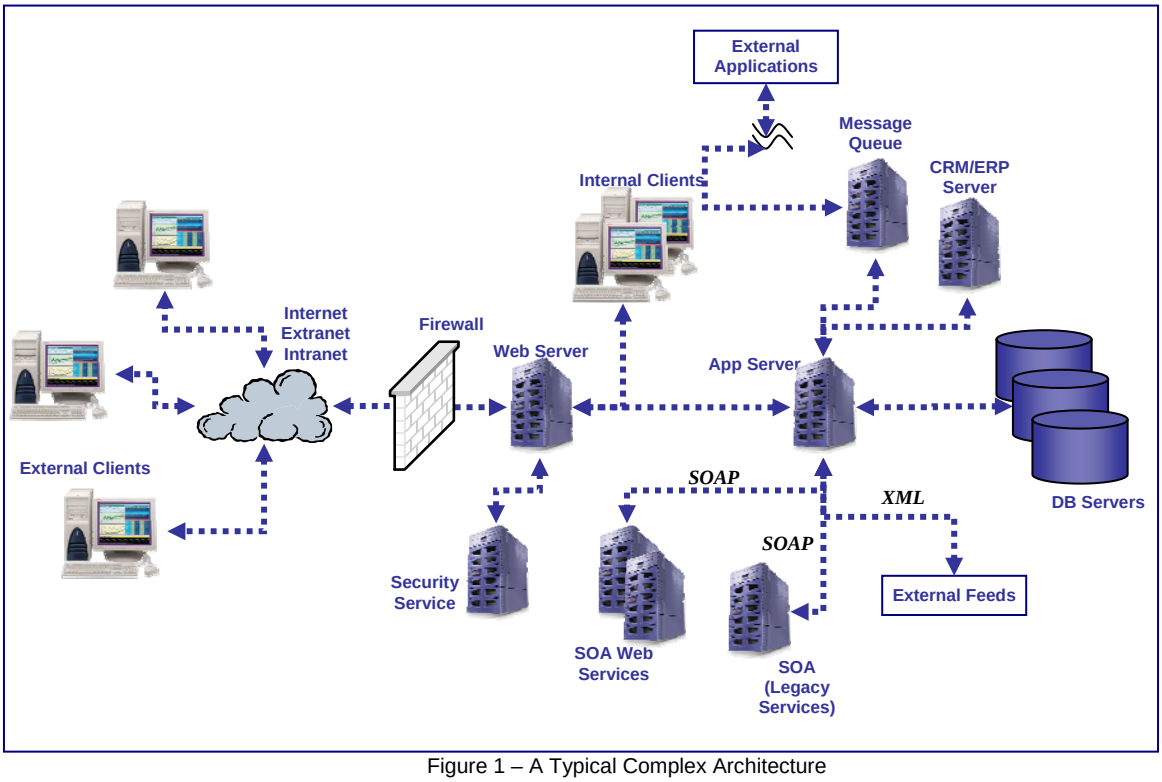


Figure 1 – A Typical Complex Architecture

For both functional and scalability issues, industry-standard testing practices, which evolved largely in response to quality issues facing the client-server architecture, have centered either on the front end (functional) or the back end (scalability/performance). This 'division of labor' derived largely from the fact that the classic client-server architecture, as a 2-tier structure, is well-covered by the approach: in the standard client-server arrangement, issues are either on the client side or on the database side. The current trend toward architectural complexity makes this simple testing scheme obsolete.

The increase in complexity, overlaid with the problems of integrating legacy and cutting-edge development, can make characterization, analysis and localization of software and system issues (functional and scalability/performance issues) major challenges in development and delivery of quality software systems. The answer to all of these questions is in the Greybox approach – by thinking inside the Grey Box, the quality of the whole can be improved at every level: component by component, subsystem by subsystem. Greybox testing promises better software quality by combining a broad conceptual approach supported by the range of testing and monitoring tools in the market with a best-practice-based engineering approach to quality.

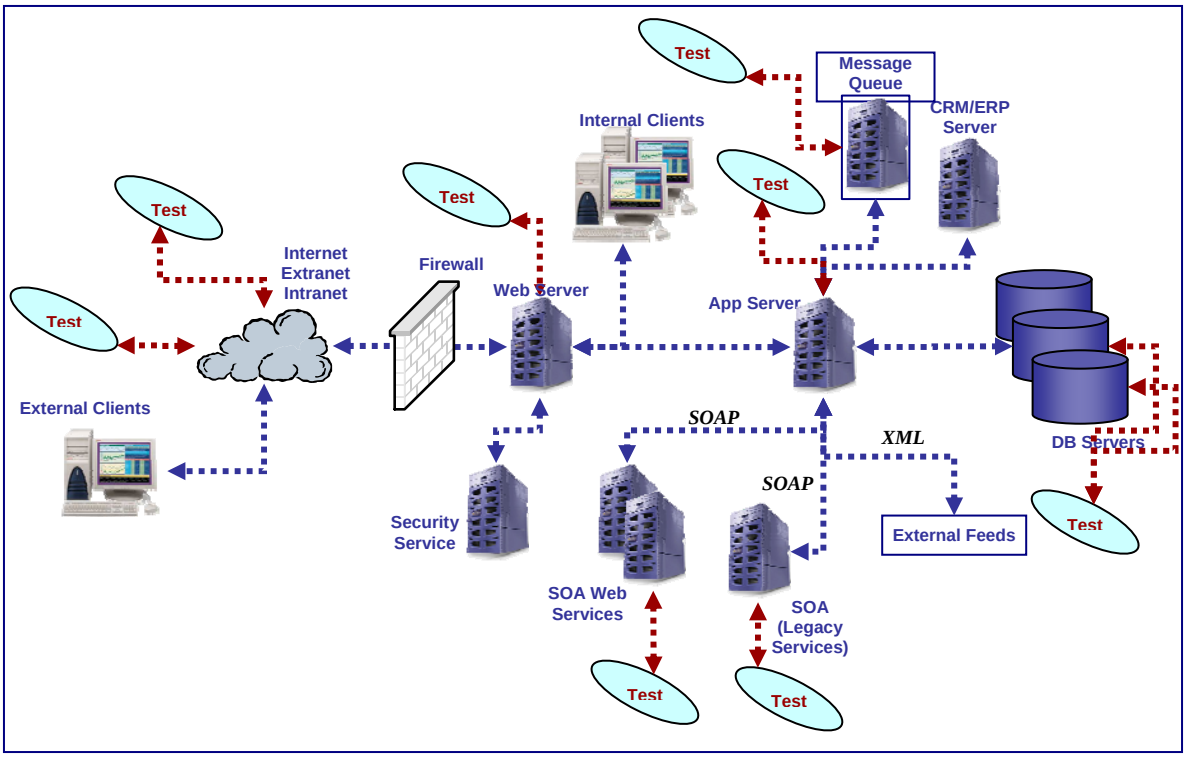
An overall quality strategy

The Greybox approach aids in determining the strengths and weaknesses of the architecture, and in pinpointing which components must be involved in resolving the functional/regression and performance/scalability-related issues of the architecture. For every question that can be asked about how an environment scales or performs, a test can be created to answer that question. In the performance/scalability arena, issues that commonly arise include:

- How many users can access the system simultaneously and still maintain acceptable response time?
- Will my high-availability architecture work as designed?
- What will happen if we add a new application or update the one I currently use?
- What should the configuration be to support the number of users we expect at launch, in 6 months and in 1 year?
- We only have partial functionality – is the design sound?

In the area of functional/regression data testing, the Greybox framework supports full data integrity validation. Because data in the enterprise may derive from diverse sources, the validation of data integrity across the full architecture is becoming increasingly critical. Assessing data integrity (including any functional transformations of data that occur during processing) both within components and across component boundaries leads to the efficient localization of each potential defect, making the tasks of system integration and defect isolation part of the standard process of development.

For questions pertaining to application functionality (UI presentation, business logic, component data handling, system integration and system data integrity), Greybox offers an analogous set of testing strategies. Whether testing the system as an assembly or components, or as a whole, test targeting every functionality-based question can be designed and executed. The Greybox approach to architecture testing exercises and analyzes the environment from the ground up.



Memory and Metrics

Greybox testing borrows from whitebox testing a concern for testing metrics gathered against the software code. This means that a Greybox implementation should, for example, utilize code coverage tools for precise measurement of what is tested during each cycle. Code coverage should be used to evaluate all testing, from the unit tests against individual components, to subsystem tests against assemblies of components to tests against the full architecture. In addition, the Greybox approach should employ memory-based tools, which offer memory corruption detection, memory leak detection, and performance profiling.

Build-Test-Report

An emerging area in the market is that of build automation, also called software production automation. Tools in this area automate the build, test and deployment processes, so that problems in the build cycle can be avoided, standard automated regression tests can be run as part of the automated build process, and deployment mistakes can be surmounted. Build automation tools can bring significant efficiencies to the build process, so that automated build-regression test-reporting cycles can be run during off-hours, giving development teams rapid evaluations of their work. Tools such as these are an important part of implementing Agile- or XP-style development.

Deployment Testing

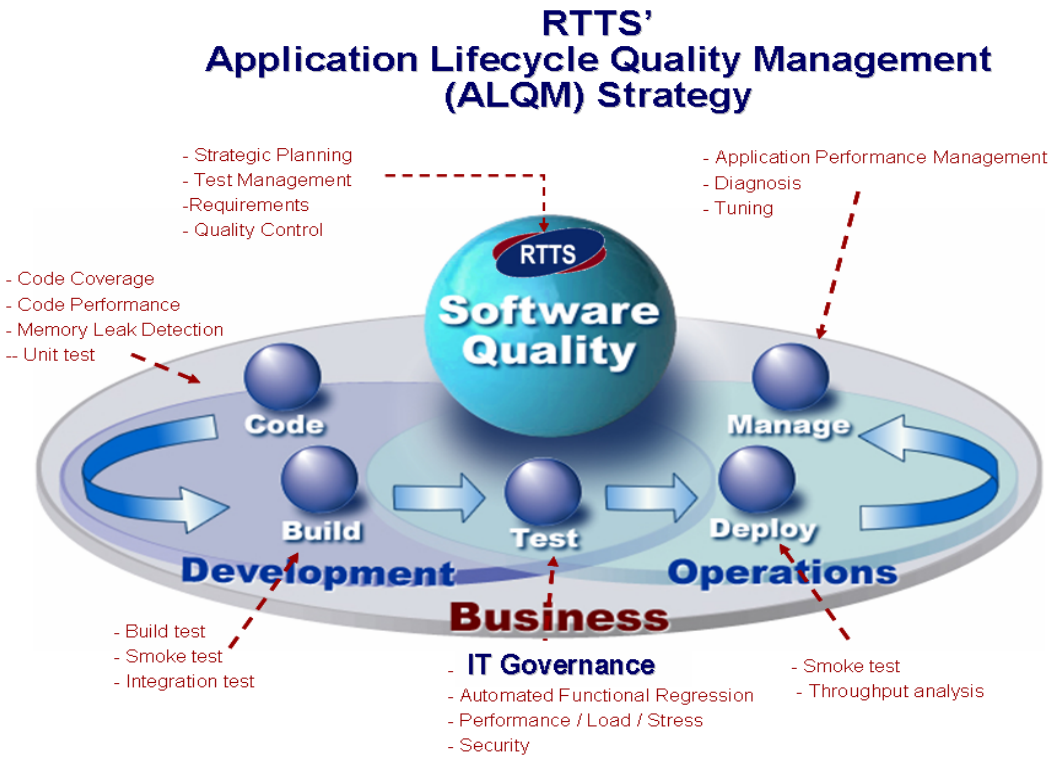
Testing performed prior to production deployment provides valuable information in preparing the finished product for the consumer, but with rare exception, testing is not performed against the production architecture. All of the artifacts created to this point can be employed in a final set of production readiness runs. The heuristics gathered at this point add to and/or reinforces all decisions made to this point regarding the final product.

Test and Monitor Tools

There are various classifications of tools employable for the testing, analysis, emulation and simulation that satisfy the implementation and monitoring requirements of Greybox testing. These tools are available both commercially and in some instances as open source. Those used in any testing instance depend upon factors such as technologies used, institutional policy, budget, available resources, skill sets and prior experience. For instance, in performance and scalability testing, the ability to generate load against a test environment might come from a single multi-protocol supported commercially purchased solution, or a mixture of commercial, open source and "home-grown" tools. Monitoring of an environment infrastructure during a test can be a feature of an existing tool or the combined use of specialized tools targeting the diverse technical components of the environment. The same testability and monitoring requirements hold true for functional testing (May want to elaborate or add). In addition to empirical test requirements, there may be instances where emulation and simulation are the only viable alternatives to obtaining heuristics in making decisions. This would include the ability to simulate real-world network conditions in a WAN-based implementation and there are tools available for accomplishing this. Finally there are modeling-based solutions that are best used once empirical data is available.

Software Testing as an Engineering Discipline

As should be clear, Greybox testing conceptualizes software testing as a sub-discipline of software engineering. Going inside the Grey box means dealing with the full range of technologies that the architecture is composed of. Each technology and component must be considered a target for testing, and the skill set to deal with a broad range of technologies is required, both from the test design and test implementation perspectives.
It is common, that enterprise architectures will combine legacy technologies along with current technologies, and a test implementation may involve SOA, SQL, AJAX, message queues, proprietary or custom protocols and more.



The skills needed to properly employ Greybox testing have grown over the years and this method requires a number of individuals collaborating and communicating in order to fully realize and successfully execute a complete assessment of a test environment. This includes but is not limited to developers, subject matter experts, networking resources, database administrators, project managers/trackers and others. Since Greybox testing employs processes from whitebox and blackbox testing methodologies, the skills of these resources also range and cover not only aptitudes in testing methodology, but also in the details of coding, monitoring, analysis and correlation.

Conclusion

The Greybox Architecture Testing strategy described here 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 and in prerelease quality evaluation to provide both diagnostic information to enhance development efficiency and a high degree of quality assurance upon release. For management of the trend toward architectural complexity and distributed computing, the Greybox strategy for architecture testing provides a robust solution.

About the authors

Bill Hayduk, founder, president and director of professional services at RTTS, has an excellent reputation in the technology field and is particularly noted for his methodology and automation expertise. Over the last 20 years, Bill has successfully implemented large-scale projects at many Fortune 500 firms. He has consulted in many business sectors including global banks, brokerage firms, multimedia conglomerates, and software, pharmaceutical and insurance companies.

Bill holds an MS degree in Computer Information Systems from the Zicklin School of Business (Baruch College) and a BA in Economics from Villanova University. He has been a selected speaker at industry-specific trade conferences, as well as a source of information for corporations and has been referenced in many industry trade publications.

Jeff Bocarsly, the functional testing division manager at RTTS, has successfully implemented automated software testing projects at many Fortune 500 firms. His experience includes project in various sectors including brokerage firms, media, pharmaceutical, and software, banking, insurance, and reinsurance companies. He specializes in implementing test automation and methodology solutions for software development groups.

Jeff holds a BS from UCLA and Masters and PhD degrees from Columbia University.

Jonathan Harris, performance and scalability division manager for RTTS, has planned and executed hundreds of performance evaluations within every major market segment. With over 19 years of experience in programming and testing, he has spearheaded the development and implementation of RTTS' proprietary scalability testing.

Previously, Jon worked as a lead testing consultant for *Promark, Inc.* where he developed the front-end interface to the *Promark Robot*, now *Compuware*'s *QA Load*. Jon studied computer science and biology at Carnegie-Mellon University.

About RTTS

RTTS is the premier professional services organization that specializes in providing software quality to critical business applications. With offices in New York, Atlanta, Philadelphia, Orlando and Phoenix, RTTS has been serving Fortune 500 and mid-sized companies since 1996. RTTS draws on its expertise utilizing its proven methodology, expert engineers and the industry's best-of breed tools to provide the foremost end-to-end solution that ensures application functionality, reliability, scalability and availability. To learn more about RTTS, visit www.rttsweb.com.