Presented at CMG 2002, December 8-13, 2002

Stepwise Refinement:
A Pragmatic Approach for Modeling Web Applications

Amy Spellmann
HyPerformix, Inc.
amy@hyperformix.com

Richard L. Gimarc
HyPerformix, Inc.
rgimarc@hyperformix.com

Stepwise refinement is a practical guide for predictive performance modeling of Web-based applications. The level of detail required in the model, and thus the time required to build the model, is driven by the performance questions to be answered. This paper demonstrates an efficient and effective modeling methodology for predicting Web application response time, evaluating scalability, and identifying and quantifying the performance improvements for application optimization alternatives.

1 Introduction

Performance modeling has proven to be a valuable tool that improves our understanding of the performance of today's multi-tier Web-based applications. Traditional analytic and statistical techniques are useful for gaining insight into the performance of individual components and their observed interaction. However, performance modeling takes the analysis to a broader level by enabling the analyst to gain additional understanding of the dynamics, dependencies, and interplay between system components and how they contribute to and affect the end-user experience.

Although the virtues of modeling are understood and accepted, there is a widespread perception that modeling takes to long. As advocates and practitioners of performance modeling, we need to begin to dispel this misconception by addressing ***The Web Performance Modeling Challenge***: How do you use performance modeling to efficiently and effectively answer Web application performance questions?

- An efficient solution is one that has minimum effort (i.e., you can quickly build a model).
- An effective solution is one that solves the problem (i.e., provides the answers you need).

One of the more challenging aspects of modeling is choosing the right level of detail to be included in a model. At one extreme, you can reproduce (i.e., rewrite) your application in the model. If your application took two years to develop, it may require a number of months to construct your model; and by the time you're done the model will probably be too late to provide the intended information. At the opposite extreme, you can represent the system as a black box and apply Little's Law to develop a model that relates population to throughput and response time. Although fast to develop, this type model delivers limited information about the interaction and dynamics of the individual components that comprise today's multi-tier Web applications.

This paper describes a methodology based on Stepwise Refinement that provides a pragmatic process for guiding the modeler in the selection of the level of detail to be included in their performance model. The level of detail is driven by the type of question to be answered by the model. By limiting the level of detail in a model, you achieve two significant benefits:

1. The model can be developed in a timely manner.
2. The model will answer the questions that are driving the modeling effort.

1.1 Analytic Versus Simulation Modeling

This paper will focus on the use of a discrete event simulation model. Simulation modeling was selected rather than analytic modeling because of the following factors:

- Constructing an analytic model of a multi-tier application is a complex task, especially when you are required to represent the various throughput-limiting components inherent in these systems.
- Web applications tend to have transient loads (bursty and varying by time of day); analytic models cannot easily capture these types of behaviors.

- Most importantly, analytic modeling cannot, without great effort, represent systems at varying levels of detail.

A simulation model is the appropriate tool for staged development and analysis. You begin with a high-level of the system, and then only add detail where appropriate. The incremental refinement of the model provides answers and insight along the way (i.e., you don't have to wait until the end), plus it supports a more focused approach to the data collection and analysis required for studying system/software scalability and application architecture.

1.2 Organization

The remainder of this paper is organized as follows:

- **Web Application Components** describes the basic components and terminology of Web applications that will be utilized in this paper.
- **Stepwise Refinement** provides an overview of the major concepts and terminology of Stepwise Refinement.
- **Level 1 – System Scalability** describes the initial level of modeling that can be used to address system scalability performance questions.
- **Level 2 – Software Scalability** describes the next level of detail which is used to address scalability issues within software subsystems
- **Level 3 – Application Optimization** describes the lowest level of detail that can be used to study optimization and tuning options for specific application components.
- **Case Study** demonstrates how Stepwise Refinement was successfully applied to a real problem.

2 Web Application Components

All Web applications can be decomposed and analyzed in a similar manner (see [LOOS2000]). As an example, consider an online brokerage. A user placing an order to purchase some stock performs the following steps:

1. Access the Home Page
2. Login to the system
3. Get a quote of the current market price
4. Place the order to buy the stock
5. Logout

This 5-page user session is illustrated in Figure 1. Each page request is referred to as a business function. During the session, the user submits a sequence of five business functions. There is user interaction time between each page request where the user reads the returned page, enters any required page input, and selects a link to step to the next page.

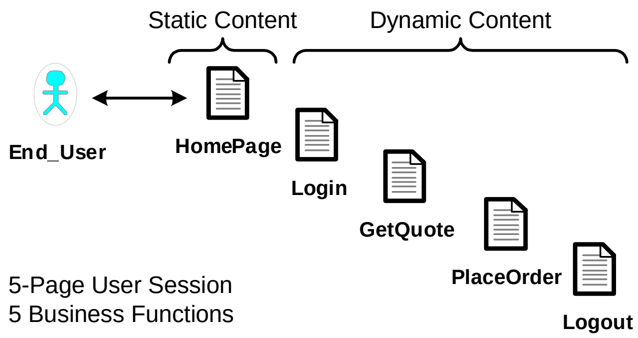

Figure 1. Business Function View of a Web Application

2.1 Page Performance Factors

The application processing required to process each page request depends on the page content. Static page content can generally be processed entirely by a Web server. Dynamic content, such as processing a login request, requires backend processing by application and database serves. Figure 2 illustrates the flow for the first two page requests in our brokerage example.

The initial Home Page request is served directly by the public web server; page content is static and can be retrieved by anyone with Web access. The Login request requires additional processing behind a protective firewall to authenticate the user and then to construct an output page that is personalized to the user.

The primary factors that determine a page's performance are dependent on the page's content:

- **Static Content.** The performance of Web pages with static content is dependent on the following three factors: page design, public Web server, and Internet performance. The page design determines the content to be displayed on the page (e.g., base page, graphic content elements). The public Web server handles all requests for page content and returns the page elements to the browser. The performance of the Internet throttles the transmission of the page elements to the browser. The primary factors affecting transmission time are latency and effective bandwidth.

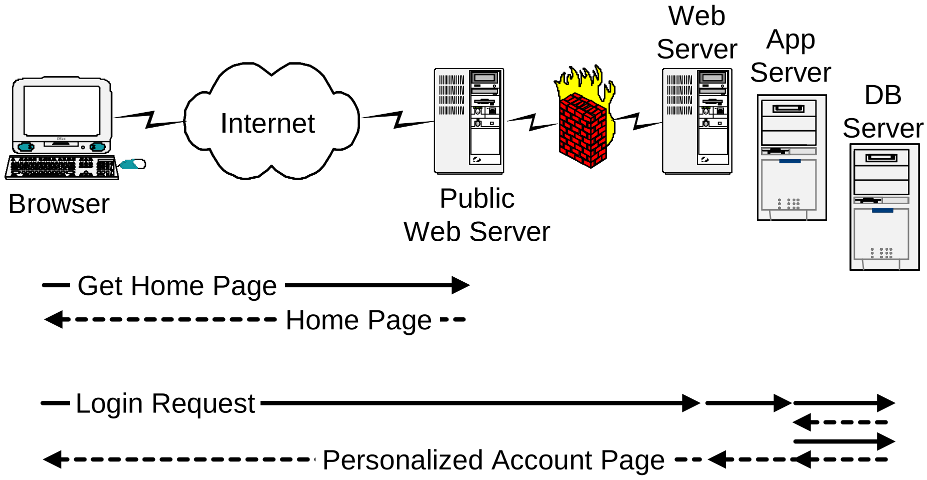

Figure 2. Web Application Flow

- **Dynamic Content.** Pages with dynamic content introduce an additional layer of processing; namely, the backend work performed by the application and database servers. The three factors that drive the performance of dynamic content pages are the following:

  1. Network components behind the firewall.
  2. Servers that are used to process the request; e.g., application and database servers.
  3. Application design; for example, application architecture and configuration, workflow between the application components, and resource usage (e.g., CPU time, memory usage).

From a high-level perspective, Web application performance is dependent on three primary factors: application design, server performance, and network performance. The difference between static and dynamic pages is the complexity of application processing. In [GIMA2000] the authors describe how to develop a performance model of a page with static content. The intent of this paper is to extend that methodology using Stepwise Refinement to examine application processing behind the firewall (i.e., Web pages with dynamic content).

2.2 Definition of Terms

Before describing Stepwise Refinement it is necessary to define three key terms that are fundamental to the methodology (see [JOHN2002] for more details).

Business Function. A *business function* represents the unit of work that is of primary interest. In most cases, a *business function* is initiated by an end-user. Business functions include those units of work that make money for the business and are generally subject to service level agreements. In a Web application, a business function can easily be mapped to a page request. A *business function* is composed of, and processed by, a set of individual *transactions*.

Transaction. *Transactions* are the basic building blocks of *business functions*; it is assumed that one or more *transactions* are required to process a *business function*. *Transactions* are defined for each tier in the execution environment and may be synchronous or asynchronous, serial or parallel so that one *transaction* may initiate zero or more child *transactions*. It is assumed that a transaction starts and stops execution within a single *software subsystem*.

Software Subsystem. A *software subsystem* refers to a software infrastructure component executing on a single server. A *software subsystem* provides the execution environment for *transactions*. The mapping of *business functions* to *transactions* to *subsystems* clearly identifies *business function flow* through the system, that is, the sequence of *transactions* required to process a *business function*.

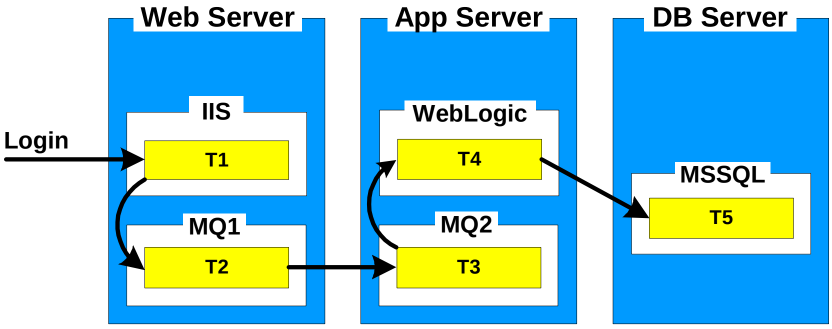

Figure 3. Login Business Function components

Figure 3 illustrates these application components for the Login business function:

- Three servers are required to process a Login request: Web, application, and database servers.
- The business function workflow starts at the Web server, uses MQSeries to pass the Login request to the application server, and the application server accesses a backend database to perform authentication and retrieve personalized content.
- Five software subsystems are utilized: two on the Web server and application server, and one on the database server.
- A single transaction executes within each subsystem to process the Login request.

It is important to note some important characteristics of this application model.

- Business functions do not directly utilize system resources. Instead, the arrival rate or pattern of the business function describes the application workload.
- Transactions are the worker bees for a business function. Transactions consume system resources (CPU, I/O, network, and memory) and determine the workflow between application components.
- Subsystems do not consume system resources. A subsystem provides the execution environment for transactions. If there are threading or other software limiting constraints, they are represented by the subsystem.
- A business function is the common thread that binds together all processing performed on its behalf. By tracing the transactions that comprise a business function, you can readily determine the subsystems and resources required for processing.

3 Stepwise Refinement

Stepwise Refinement is a process that enables performance analysts to address

The Web Performance Modeling Challenge:

How do you use performance modeling
to efficiently and effectively answer
Web application performance questions?

Two critical success factors are:

1. Focus on the key performance questions and business goals (effective).

2. Include the minimum amount of detail required to answer the questions (efficient).

It is often the case that the project's goals are not clearly stated. Your goal should not be to "build a model of my new eCommerce system". Instead, you need a more specific goal such as "determine if the planned deployment configuration can handle 2,000 customer requests per minute and satisfy our customer's service level agreement". The business unit funding the development effort will derive immediate value from answers to specific questions that address both business and technology issues.

Abstraction is the key to successful modeling. The amount of detail to include in a model is an ongoing challenge. It is often the case that the modeler has intimate knowledge and understanding of the application, and is determined to reproduce that detail in the performance model. This has been the cause for a number of failed modeling efforts. Two quotes from Adrian Cockcroft [COCK2000] emphasize the importance of selecting the *right* level of abstraction:

- "The hardest thing to learn for successful modelling is to let go of the details".
- "Be ruthless about paring down the complexity to a minimum to start with".

Stepwise Refinement guides the modeler by selecting the appropriate level of detail required to answer the project's questions.

3.1 Levels of Abstraction

The level of abstraction (i.e., amount of detail) used in a model is determined by the type of question(s) that must be answered. We propose the following three levels of abstraction:

- **System Scalability.** System Scalability addresses the end-to-end application and infrastructure capacity. Questions include: how many users can the system handle (peak), what's the maximum throughput of this system, where are the sensitive components (network links, tiers), how many servers are required at each tier?
- **Software Scalability.** Software Scalability is a special case of System Scalability where the limiting tier of the system is known (or a specific tier is targeted for performance improvements). The primary question is: what software subsystem limits the throughput of the tier? You can also use a Software Scalability model to evaluate the effect of threading and connection pool size on application performance.
- **Application Optimization.** Optimization is focused on performance tuning for applications and infrastructures. Questions addressed by Optimization studies include: how should the application be optimally configured, what is the effect of changing application-specific tuning knobs, what is the best load balancing algorithm, will new database indexes reduce response time, will caching really improve my response time, what is the effect of application chattiness, should I redesign my Web pages for performance, what is the effect of software security overhead?

Scalability is the ability of a computer application or product (hardware or software) to continue to function well as it is changed in size or volume in order to meet a user need [WHAT2002]. Scalability is the ability not only to function well in the rescaled situation, but also to actually take full advantage of it in terms of performance (where performance is the total effectiveness of a computer system, including throughput, response time, and availability). System Scalability is a high-level analysis of the entire multi-tier application. A more focused analysis on specific application components is the subject of Software Scalability: the throughput limiting software subsystems are represented in more detail to answer subsystem-specific scalability questions.

Optimization is the process of tuning an application to meet the scalability requirements. The goal of Application Optimization is to expose application tuning knobs in the model so that application-specific behavior can be observed, understood, and tuned.

Each Stepwise Refinement level has a specific focus:

- **System Scalability** - What is the network component or tier that limits throughput or performance?
- **Software Scalability** - What is the software subsystem that limits throughput or performance?
- **Application Optimization** - What is the application resource that limits throughput or performance?

The amount of detail required in a performance model increases as the questions progress from System Scalability to Software Scalability to Application Optimization (see Figure 4). A microscope analogy is useful for understanding. System Scalability is the 50,000 foot view. As you adjust the magnification control to get a closer view of specific components of the application, your field of vision shrinks; you see detail in the focused component, but your view of the surrounding area decreases.

3.2 Applying Stepwise Refinement

In general, the steps in a performance modeling project are:

1. Identify the specific goals and performance questions. This step determines your measure of success; your modeling project will be successful if you satisfy the goals and provide timely answers to the performance questions.

2. Determine the appropriate level of abstraction required to answer the project's questions.

3. Collect measurement data at a granularity matching the abstraction level.

4. Construct the performance model; the level of detail in the model matches the abstraction level.

5. Develop model input parameters from the collected measurement data.

6. Evaluate the model to provide answers to the project's performance questions.

Stepwise Refinement, guides step 2 above and directs data collection and model construction. Once the appropriate abstraction level has been selected, data collection and model construction may begin. Both of these tasks are performed incrementally, progressing from level to level.

Figure 5 illustrates how Stepwise Refinement is applied to model construction. You always begin with a System Scalability model. It is often the case that a System Scalability model will provide the answers to your project's performance questions. If you need further detail to answer the questions, you refine the System Scalability model to represent the software subsystems of interest. This process continues in a similar manner for the Application Optimization model. The advantage of this approach is two-fold:

1. You may be able to answer the project's questions with a less detailed model (i.e., the modeling project is completed in less time).

2. All work can be leveraged as you refine your models into lower levels of abstraction. The System Scalability model is the foundation for the Software Scalability model. Likewise, the Software Scalability model is the foundation for the Application Optimization model. In both cases, you are simply drilling down and refining the model's representation of specific software subsystems. The model's representation of the surrounding system components and subsystems is invariant and may be readily reused.

Data collection applied to the three Stepwise Refinement levels is also repeatable and reusable. Figure 6 illustrates the major data collection tasks. The categories of information/data to be collected are as follows:

- **Application.** This application's key business functions are identified and their workflow is determined.
- **Execution Environment.** The servers that

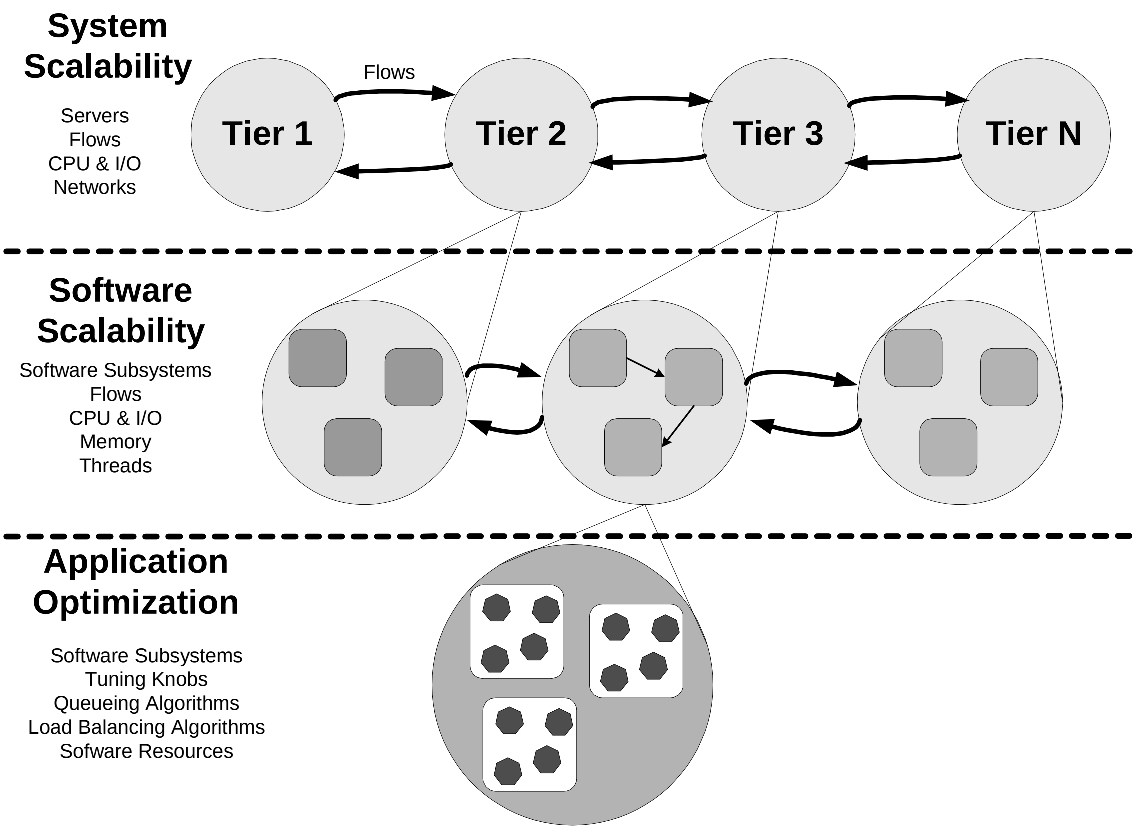

Figure 4. Stepwise Refinement - 3 Levels of Abstraction

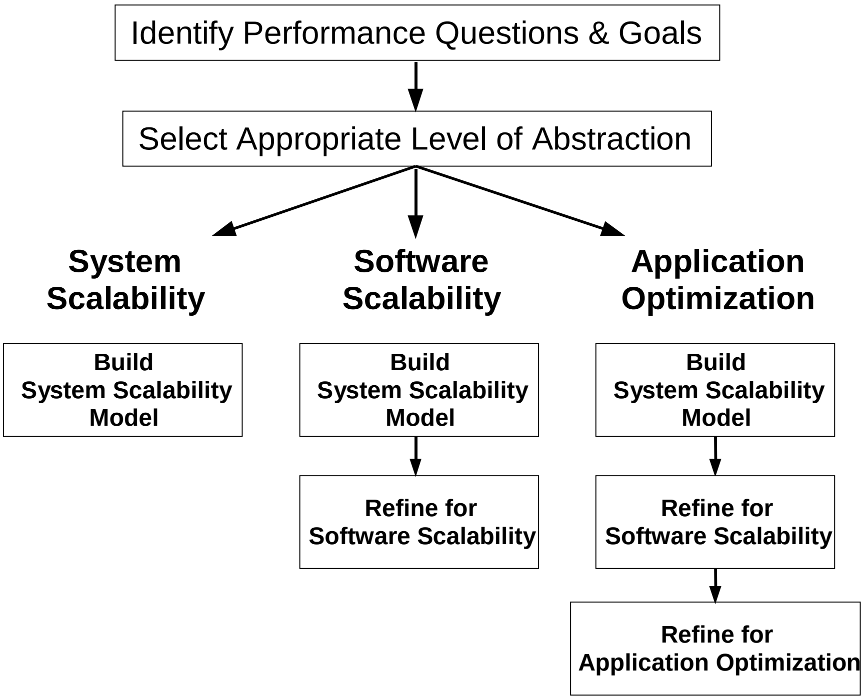

Figure 5. Applying Stepwise Refinement

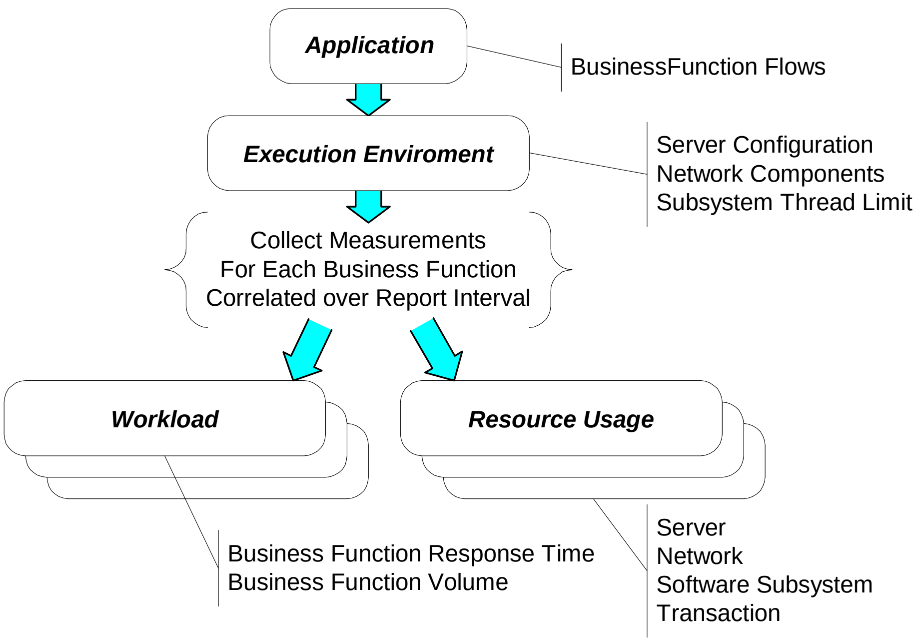

Figure 6. Data Collection Tasks

will be hosting the application are identified. The network topology is determined along with any software subsystem threading limitations.

- **Workload.** The workload describes the arrival rate and pattern of the application's business functions. You may also have access to business function response times that can serve as model validation targets.
- **Resource Usage.** The resource usage of each transaction, subsystem, server, and network segment that actively participates in processing a business function must be determined (by measurement, inference, or estimation).

Some of the collected data can be readily reused for any of the three Stepwise Refinement levels: Application, Execution Environment, and Workload. The difference between the three levels is in its representation of software subsystems. As detail is added to software subsystems, there is an accompanying increase in the amount of transactions. A single software subsystem can represent all of the processes running on a server, a related group of processes running on a single server, or a single process.

4 Level 1 – System Scalability

A System Scalability model is always the initial step in a performance modeling study. If your plan is to develop a lower-level Application Optimization model, you still begin with a System Scalability model. System Scalability data is required no matter what modeling level is needed to answer the project's questions.

System Scalability addresses the end-to-end application and infrastructure capacity. Typical questions that may be addressed by System Scalability are the following:

- Will the planned configuration support expected growth?
- At what workload does the customer response time fail to meet performance requirements for this application?
- What portion of the system will fail (bottleneck) and at what workload?
- What is the capacity of the planned system?
- What size server(s) should be used to support this application?
- What network bandwidth is required to support this application?
- Which system components are most sensitive to increasing user counts and transaction rates?

4.1 System Scalability - Abstraction

A System Scalability model uses a high-level of abstraction: one subsystem per server and no application specific processing. An example for our Login business function is shown in Figure 7:

- Each server contains a single software subsystem.
- There is a single transaction that runs on each subsystem.
- Business function processing is specified by transaction-to-transaction workflow.
- Resource requirements are specified at a server level: CPU, I/O, and network traffic.

This representation is a simplification of the actual processing. In order to gain an initial understanding of the system under study, it is best to start with a high-level model and add detail where necessary.

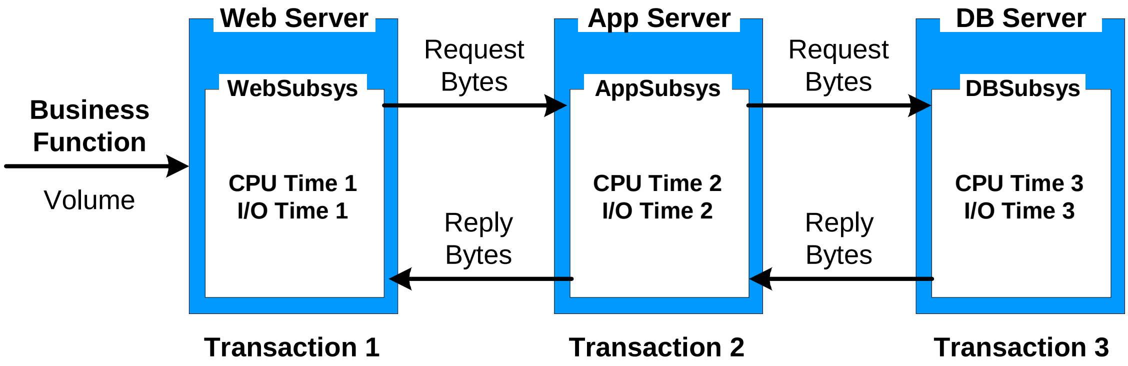

Figure 7. System Scalability - Abstraction

4.2 System Scalability – Data Requirements

The data requirements for a System Scalability model are listed in Table 1 below. The table lists the major application components (rows) and the categories of data/information to be collected for each (columns). Application data identifies the software application components. Execution environment lists data elements that are invariant and may be reused if the System Scalability model is a stepping-stone to a more detailed Software Scalability model. The workload data required to construct the model includes the number of users, think time, arrival rate, and response time. Resource usage includes CPU utilization and CPU per transaction. Some of the data elements may be used to validate the model; for example, business function response time and server-level CPU utilization and I/O rate.

4.3 System Scalability – Data Collection

Data collection for System Scalability is straightforward. The execution environment can be determined from interviews or examining the actual system components. Business function flow diagrams can also be determined from interviews or application documentation, but a more rigorous approach is to analyze the results from a network packet trace of each individual business function.

Workload and resource usage data can be collected from most system built-in monitors. Resource data is required at the highest level, so system-level CPU utilization and I/O measurements are satisfactory. Analyzing logs generated by the Web server can determine workload information for a Web-based application.

Validation data for the application's servers are readily available from standard measurement tools. However, business function response time is a challenging data element to capture. In a load test environment the workload driver can capture this metric.

Note that measurements for a System Scalability model are at the system level; no process-level data is required.

5 Level 2 – Software Scalability

A Software Scalability model is a special case of System Scalability. In this case, the limiting tier of a system is known (from the System Scalability study) and is targeted for more detailed analysis and potential scalability improvements. The effort already spent to develop the System Scalability model will be leveraged in the development of this more detailed Software Scalability model.

Software Scalability addresses the throughput limiting characteristics of a software subsystem. Typical questions that may be addressed by Software Scalability are the following:

- How should software/applications be partitioned to maximize the use of servers?

	Application	Execution Environment	Workload	Resource Usage
Business Function	Bus Fn Name Flow Diagram		Number of Users Think Time Arrival Rate Response Time	
Transaction	Trans Name Bus Fn Name Subsystem Name			CPU per Trans I/Os per Trans I/O Time Request Bytes Reply Bytes
Software Subsystem	Subsystem Name	Hardware Name Thread Count		
Hardware		Hardware Name Number of CPUs CPU Speed Number of Disks Disk Capacity		CPU Utilization Disk Utilization I/O Rate
Network		Network Name Topology Diagram Protocol Bandwidth		Average Latency Network Utilization

Table 1. System Scalability - Data Requirements

- For a given input rate, what application functions can be enabled while still providing the required level of service?
- How many middleware "pipes" do I need to support peak workload?
- How can applications be redeployed to support server consolidation?

5.1 Software Scalability - Abstraction

Software Scalability refines the abstraction used for System Scalability to provide more detail for application-limiting software components. The refinement is at the subsystem level. It is assumed that the software subsystem limiting performance is decomposed into two or more subsystems. As new subsystems are introduced, there is an accompanying increase in the number of transactions being modeled. As in the System Scalability model, no application-specific processing is represented in a Software Scalability model.

Figure 8 shows a portion of the Login business function. It is assumed that we are drilling down on the Web server and application server tiers. This figure contrasts with the System Scalability abstraction shown in Figure 7 in the following ways:

- Two software subsystems are represented on the Web and application servers.
- There are two new transactions introduced: one for each of the new subsystems.
- Business function flow is more detailed and includes the transaction-to-transaction workflow through all software subsystems.
- Resource requirements for each of the subsystems now must be collected at the process level (rather than the server level for System Scalability).

The level of detail increases with a more refined representation of individual software subsystems. There is a corresponding increase in the complexity of

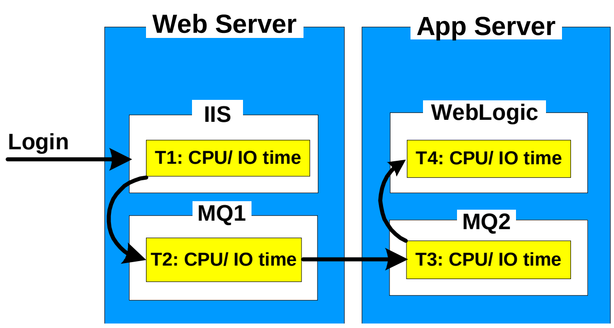

Figure 8. Software Scalability - Abstraction

the business function flow, the number of transactions represented in the model, and the granularity of the data required to characterize transaction processing within each subsystem.

5.2 Software Scalability – Data Requirements

The data requirements for a Software Scalability model are listed in Table 2. The non-shaded entries represent the new data required by Software Scalability; shaded entries represent the data required in the earlier level, System Scalability.

The introduction of new subsystems into the model requires a refinement to the business function flow; flow must now be refined to include the new subsystem-to-subsystem flow. Accompanying the introduction of new subsystems is new transactions (at least one per subsystem). Since subsystem performance is now being studied in more detail, it may be necessary to perform model validation at the subsystem level. All other data elements that were collected from the System Scalability may be reused.

5.3 Software Scalability – Data Collection

Data collection changes for Software Scalability focus on the newly introduced software subsystems: if a single subsystem is decomposed into two or more subsystems, then business function flow must be refined and measurement data must be collected (or estimated) that describes a transaction's resource usage within the new subsystems.

In general, subsystem (or process) measurements are a challenge to collect. Our goal is to populate Table 2 with resource usage for each transaction. Mainframe measurement tools report these metrics for the major subsystems (e.g., CICS and DB2). Collecting these measurements on Unix and Windows servers is more problematic.

6 Level 3 – Application Optimization

Application Optimization follows as the next logical step from Software Scalability; the problem has been identified, what is the most cost-effective fix? Optimization focuses on the evaluation of tuning alternatives and quantification of the expected improvements for each. For Optimization questions, we again refine the software subsystems. However, this time we add application-specific details such as connection pools, load balancing algorithms, JVMs,

	Application	Execution Environment	Workload	Resource Usage
Business Function	Bus Fn Name Flow Diagram		Number of Users Think Time Arrival Rate Response Time	
Transaction	Trans Name Bus Fn Name Subsystem Name			CPU per Trans I/Os per Trans I/O Time Request Bytes Reply Bytes
Software Subsystem	Subsystem Name	Hardware Name Thread Count		CPU Utilization Disk Utilization I/O Rate
Hardware		Hardware Name Number of CPUs CPU Speed Number of Disks Disk Capacity		CPU Utilization Disk Utilization I/O Rate
Network		Network Name Topology Diagram Protocol Bandwidth		Average Latency Network Utilization

Table 2. Software Scalability – Data Requirements

	Application	Execution Environment	Workload	Resource Usage
Business Function	Bus Fn Name Flow Diagram		Number of Users Think Time Arrival Rate Response Time	
Transaction	Trans Name Bus Fn Name Subsystem Name			CPU per Trans I/Os per Trans I/O Time Request Bytes Reply Bytes Logical I/O Count Write Percent
Software Subsystem	Subsystem Name	Hardware Name Thread Count		CPU Utilization Disk Utilization I/O Rate Cache Hit Percent Locking Factor
Hardware		Hardware Name Number of CPUs CPU Speed Number of Disks Disk Capacity		CPU Utilization Disk Utilization I/O Rate
Network		Network Name Topology Diagram Protocol Bandwidth		Average Latency Network Utilization

Table 3. Application Optimization – Data Requirements

containers, database tables and indexes, disk/data partitioning, etc. The effort already spent to develop scalability models will be leveraged in the development of this more detailed Application Optimization model.

Application Optimization is focused on performance tuning for specific applications and software subsystem components. Typical questions that may be addressed by Application Optimization are the following:

- How much memory should I add to reduce the response time of my database application?
- What improvement can I expect by increasing the batch size for my message queuing middleware?
- How will my response time be improved by adding a new index to my database?
- Can I increase my workload without having to clone a new JVM in my application server?

6.1 Application Optimization - Abstraction

Application Optimization refines the generic software subsystem abstraction used in Software Scalability to contain a more true-to-life representation of an application. The refinement is at the subsystem and transaction level. Specific assumptions about the Optimization abstraction level are the following:

- Subsystems have application specific behavior, resources, and tuning knobs.
- Transactions have application specific resource requirements.

Figure 9 shows a refinement of the database for our Login business function. It is assumed that we are constructing a more detailed model that is better able to represent the internal operation of a database. This figure differs from the Software Scalability diagram by

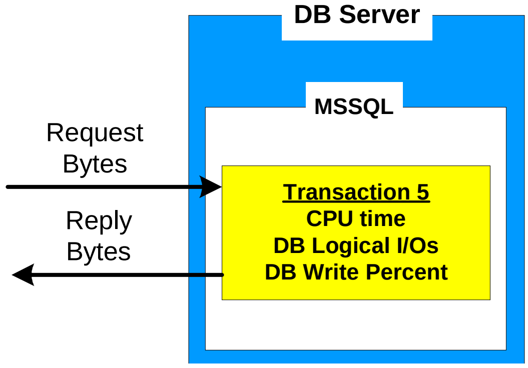

Figure 9. Application Optimization - Abstraction

the addition of database specific processing characteristics.

It is important to note that Application Optimization is the only Stepwise Refinement level that contains application-specific processing. The two scalability abstraction levels represented application components at a higher level; they did not contain any application-specific processing.

6.2 Application Optimization – Data Requirements

The data requirements for our Application Optimization model are listed in Table 3. The non-shaded entries represent the new data required by Application Optimization; shaded entries represent the data required and available from the earlier level, Software Scalability.

The use of a database specific subsystem requires two new data elements for both the database subsystem and the transactions that run on the database. The data elements shown in the table are examples of the types of new data requirements that are expected for an Optimization model.

6.3 Application Optimization – Data Collection

Data collection for Application Optimization requires the use of an application-specific monitor. For our database example, the required measurements from Microsoft SQL Server can be obtained directly from the extensions to Performance Monitor and SQL Server's Query Analyzer and Profiler.

7 Case Study

This section contains a case study describing the application of Stepwise Refinement to an actual modeling project. The subject of this study was an organization that was preparing to deploy a new collaborative eBusiness application. The application was in the test stage of the development cycle, and load testing proving to be very expensive and limiting. Thus, a performance model was needed to expand the scope of the load test and predict scalability of the eBusiness application.

The business goals of this study were the following:

- Demonstrate the accuracy and effectiveness of modeling.
- Predict capacity of critical application components under increasing loads.

- Predict end-to-end performance for the defined business function under increasing loads.

If this project were successful, then modeling would help the organization focus their scarce load testing dollars. That is, modeling would increase the coverage of the load testing; accurate predictive modeling would be used in place of actual load testing for cost-prohibitive test scenarios.

The specific performance questions to be answered by this project were the following:

- Does the system scale as the number of users and services increases?
- What system components are most sensitive to increasing user counts and transaction rates?
- What are the component capacity constraints of the system?

Choosing the Right Level of Abstraction. The project questions matched those that could be answered with a System Scalability model. Our initial project discussions revealed that there were a number of commercial software packages being used in this application (e.g., Oracle and a document repository). Could our System Scalability model answer the questions without having to add detailed application-specific representations of the database, document store, and document management software?

Project Delivery Strategy. The following strategy was used for the delivery of the project:

- Design Project Approach. Initial project planning identified the key business goals and performance questions to be answered. Knowing these, we selected the appropriate level of abstraction for the model (System Scalability).
- Conduct Performance Tests. Since the application was still in the test lab, we were able to perform three performance tests that provided the data for our System Scalability model:

    1. Business function flow was determined by analyzing the output from a network packet trace.

    2. A load test was performed for each business function to determine the resource usage at each server in the system.

    3. A second set of load tests was performed to generate validation targets. Recall that one of the goals was to demonstrate the accuracy of modeling; these load tests served as our modeling target.

- Develop Model Input Parameters. The model's input parameters and topology was developed using the following information:

    1. Business function arrival rates were obtained from the load test driver.

    2. Transaction resource usage at each server was derived using `vmstat` and `sar` data that was collected during the load tests.

    3. The network and hardware topology were described in system design documents.

- Construct the Model. A high-level System Scalability model was constructed. Model parameters describing the execution environment, application, workload, and resource usage were assigned and populated.

- Evaluate Scalability Scenarios. The model was then used to evaluate performance as the user load on the system increased incrementally from 10 users up to 200 users.

System Topology. The system topology is shown in Figure 10. Users accessed the system from a Web browser across the Internet. There were six servers that hosted the application. The *bubbles* attached to each server represent the software subsystems (where there was one subsystem per server).

7.1 Data Requirements

The data requirements were precisely those listed earlier in Table 1 for a System Scalability model.

Business Function. A flow diagram was developed for each business function by analyzing a network packet trace. In addition to determining flow, the packet trace also provided measurements for transaction Request and Reply Bytes. The number of users and think time were obtained from the load test scripts. Validation data included both business function response time and throughput.

Transaction. Transactions were modeled with CPU and network traffic. CPU time per transaction was computed using the `vmstat` and `sar` data collected at each server during the load tests.

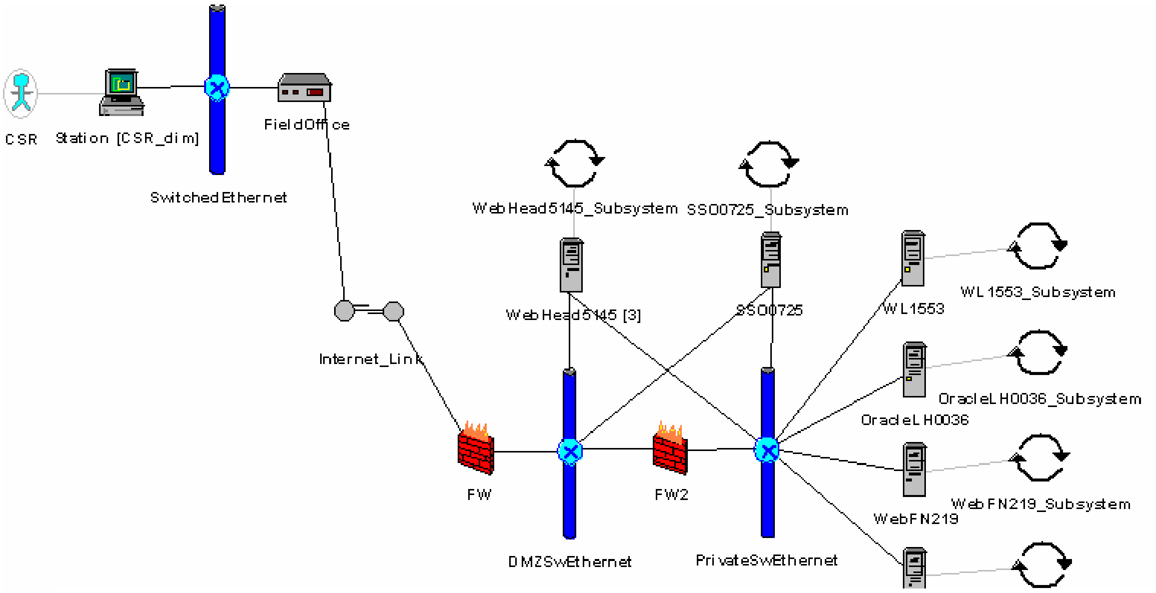

Figure 10. Case Study - System Topology

Software Subsystem. It was assumed that a single software subsystem ran on each server. For this initial model, thread constraints were not used.

Hardware. Each server was modeled using only CPU (i.e., I/O was not used in the initial model). Server-level CPU utilization was used as one of the validation targets.

Network. The network topology and description was determined from design and test documents. Since the load test was being performed over the Internet, network latency and bandwidth were used to describe the connectivity of the load driver to the system under test (the Internet_Link icon in Figure 10).

7.2 Model Results

The model was used to evaluate business function performance as the number of users was increased from 10 up to 200. The graphs in Figures 11, 12, and 13 show the results:

Database Utilization. Figure 11 shows the utilization of the busiest server, the database. The measured line represents what was observed during actual load test; the modeled bars are the predictions made by the model. Two observations can be made:

- The model accurately predicts server utilization over all loads.
- Something unusual is happening to the system at about 50 users that prevents the database server from reaching a higher utilization.

Response Time. Figure 12 shows business function response time plotted against user load. This is the typical type of curve we expect to see; response time increases with user load.

- The model accurately predicts business function response time over all loads.
- Unlike the database utilization graph, response time continues to increase with the load. There is likely a bottleneck in the system that is affecting end user response time but throttling the load offered to the database server.

Business Function Throughput. Figure 13 shows business function throughput as a function of load. This curve has a similar shape as the database utilization; as the load increases there is a flattening out of throughput. The line labeled Internet Link was determined to be the culprit; the Internet link between the load driver and the system under test was the bottleneck that was limiting throughput. Two observations from the throughput figure:

- The model accurately predicts business function response time over all loads.
- The throughput graph resembles the database utilization graph; as the offered load increases, the workload arriving at downstream components is throttled similar to the rate of work leaving the system.

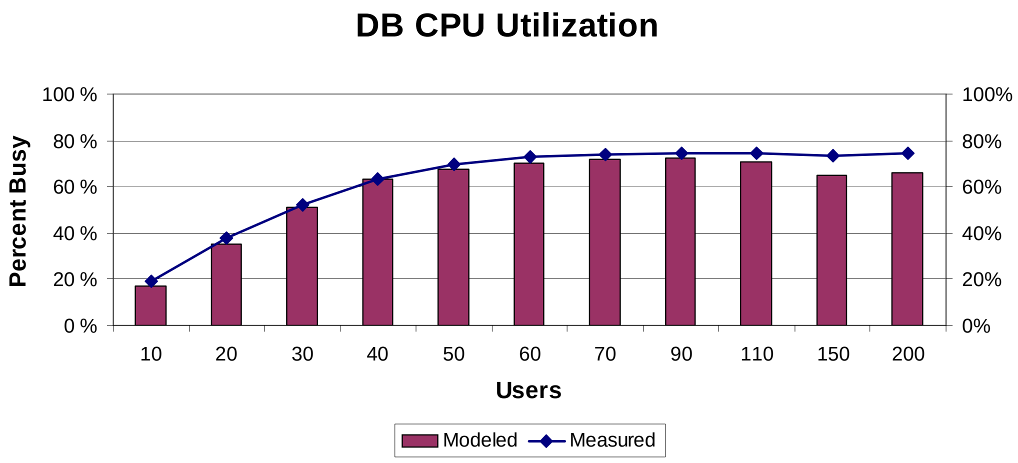

Figure 11. Case Study - Database CPU Utilization

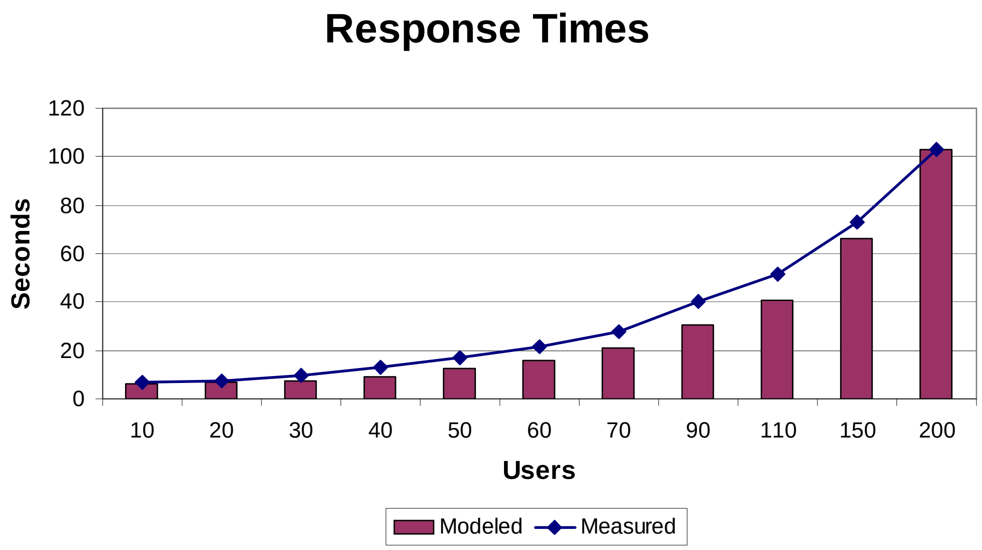

Figure 12. Case Study – Business Function Response Time

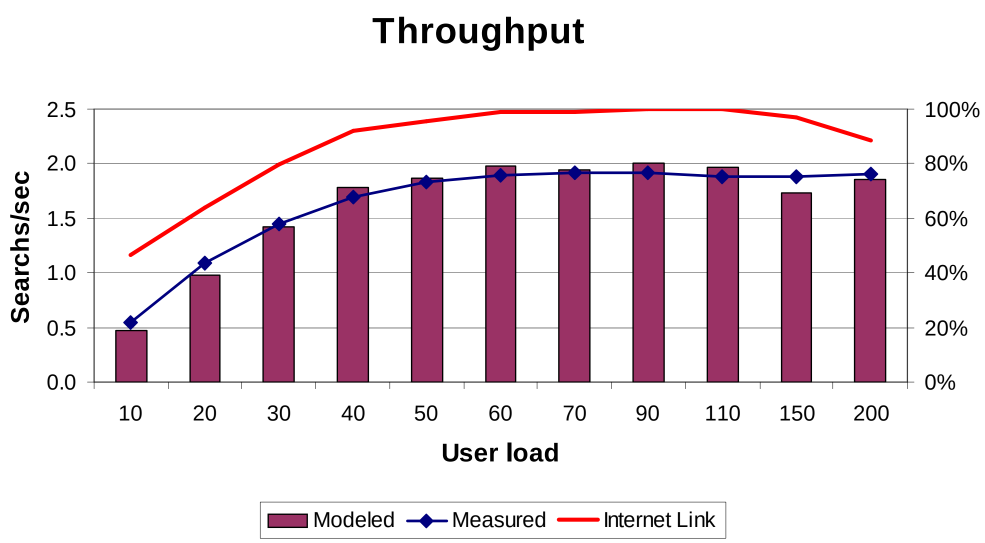

Figure 13. Case Study - Business Function Throughput

7.3 *Case Study Summary*

The accuracy of performance modeling was clearly demonstrated. Furthermore, the level of detail included in the System Scalability model was minimal, which improved the time required for model development (one week). Referring back to the project's goals and performance questions, we can confidently state the following:

- The model tracks well with measurements over all loads.
- Network limitation impacted load test (questionable results above 50 users).
- The Oracle database can't keep up with the load (busiest server).
- Modeling expanded the scope of the load test by removing the network bottleneck (remove the network bottleneck from the model and repeat the load test).

8 Summary

This paper has described Stepwise Refinement, a methodology that provides a pragmatic process for guiding the modeler in the selection of the level of detail to be included in their performance model. The level of detail is driven by the type of question to be answered by the model. By limiting the level of detail in a model, you achieve two significant benefits:

1. The model can be developed in a timely manner.
2. The model will answer the questions that are driving the modeling effort.

Performance modeling is a valuable tool that can provide immense value to business. The challenge facing the practitioners has been, "how do you do it quickly". Stepwise Refinement provides the guidance needed to determine the appropriate level of detail needed in a model. A high-level System Scalability model can be built rapidly, and may provide the required answers. If not, then it serves as a well-formed foundation for more detailed Software Scalability or Application Optimization.

Based on our experience, we have seen that about 75% of modeling projects could have been satisfied with a System Scalability model. Another 20% required more detailed Software Scalability, and a final 5% demanded a drill-down to Application Optimization.

Quantifying application scalability and identifying bottlenecks is prime hunting ground for performance models. Scalability models require minimal knowledge of the internals of the application being studied, and can be created and evaluated in days rather than months.

One of the overriding goals of Stepwise Refinement is to transform modeling from an art form limited to a small set of practitioners, into a process that can be used efficiently and effectively to address today's performance problems.

9 References

[COCK2000] Adrian Cockcroft, "Scenario Planning – Part I", Sun BluePrints OnLine, Feb. 2000

[GIMA2000] Richard L. Gimarc and Amy Spellmann, "Performance Modeling for Web Application Optimization," CMG 2001 International Conference.

[JOHN2002] Adrian Johnson, Amy Spellmann, and Richard Gimarc, "How do you eat an elephant? or, How to digest application performance in bite-sized chunks", CMG 2002 International Conference.

[LOOS2000] Chris Loosley, Richard L. Gimarc, and Amy C. Spellmann, "E-Commerce Response Time: A Reference Model", CMG 2000 International Conference.

[WHAT2002] http://www.whatis.com