Performance
Requirements
Gathering &
Analysis

This document presents checklists for performance requirements gathering during IT Systems planning, and thereafter provides techniques for performance requirements analysis. Qualitative analysis is illustrated through a set of ten realistic examples. Quantitative analysis is provided to show how missing requirements can be derived or how requirements can be validated.

Copyright (C) 2011 Rajesh Mansharamani
Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts.

A copy of the license is included in the section entitled "GNU Free Documentation License".

http://www.gnu.org/licenses/fdl.html
With increase in size, complexity, and integration of business applications, and increasing end user expectations of responsiveness and quality of service, the IT organization is faced with the big challenge of managing system performance. At the same time, executive management expects IT departments to maximize the return on investments already made in the hardware, software, and networks. The challenge has become intense with the rising expectations of both end users and the management. Thus it is imperative to have clear expectations on performance right at the outset.

Performance engineering should be an integral part of the software development lifecycle. It should start as early as the requirements analysis phase. This chapter describes a methodology for Performance Requirements Analysis. The power of performance requirements analysis done well is illustrated through several case studies of real life examples, both through qualitative patterns and through quantitative analysis.

Section 1 of this document describes the performance requirements gathering phase and several checklists are provided for project management purposes. Section 2 illustrates the value of qualitative requirements analysis for performance and culminates with a performance requirements analysis matrix to aid the software practitioner in mapping requirements to the right analysis pattern. Section 3 uses simple performance modelling techniques to quantitatively validate and analyse performance requirements, and the value of the same is again illustrated through real life examples.

1. Requirements Gathering for Performance

Gathering of requirements for performance comes in two categories. Firstly, studying and understanding specifics of the workload, and secondly, gathering performance targets. By workload we mean the load on the system in terms of number of transactions that need to be simultaneously processed or the amount of processing that is required to be done. Performance targets are for the basic performance quantities of response time, throughput, and resource utilization.

The study of the workload involves performing a usage profile of the system, or more loosely studying the ‘load’ for which the system needs to ‘work’. Inputs are to be collected for six areas:

- Business transactions for online processing
- Batch processing
- Reports processing
- Business data volumes
- User interaction
- External interfaces for the business applications
1.1 Workload Gathering for Online Transaction Processing (OLTP)

Table 1 provides a checklist for gathering inputs on the workload for online transaction processing (OLTP), in your system.

**Table 1: Workload Inputs to be gathered for OLTP**

| 1. The list of business components of the system |
| 2. For each business component, the transaction processing workload (including queries): |
| a. The list of online processing transactions |
| b. The list of steps for each online transaction (called elementary transactions) |
| c. The ‘mix’ of transactions in terms of percentages and/or dependencies |
| d. Segregation of transactions by criticality |
| e. Separation into synchronous and asynchronous transactions |
| f. Derived transactions that emanate out of workflow requirements |
| g. The average transaction rates, as fine-grained as possible. For example, transactions per second, transactions per minute, transactions per hour, transactions per day, transactions per week, etc. |
| h. The peak transaction rates |
| i. The estimated growth in transactions |
| j. The working hours of the business, and the periods of maximum transaction load per type of transaction |
| k. The number of web or client/server interactions per transaction |
| l. The transaction complexity in terms of: |
| i. Number of screen interactions |
| ii. Communication protocol of screen interactions (for example, HTTP, Oracle NCA, TCP/IP) |
| iii. Number of fields, list boxes per screen |
| iv. Amount of data fetched per screen, number of records |
| v. Number of records processed by the transaction |
| vi. Relative complexity definition (for example, transaction A is simple since it has two screen interactions with 3 fields each, transaction B is moderate since it has 5 screen interactions with 2 to 5 fields each, and transaction C is complex since it has 10 screen interactions with 10 fields each) |
| vii. Amount of data (in bytes) sent and returned per screen |

Some points stated in this checklist are detailed below. These are:

1. Transaction criticality
2. Synchronous processing
3. Workflow processing

Transaction criticality is usually tagged as one within one of three categories:

- **Mission critical**, which require not only timely processing but full business continuity. Any delay or loss of such transactions incurs a very significant cost to the business or in certain missions can be critical with respect to safety. For example, a stock exchange trade is a mission critical transaction since if it is lost by the IT system it can incur a very high cost penalty by a regulatory body. A control message sent to an online controller of a train to maintain safe breaking distance is also a mission critical transaction because
it needs to reach in a deterministic amount of time and its loss can be critical to the safety of passengers in the train.

- **Enterprise critical**, which require timely processing and close to 24x7 availability, but where small disruptions can be tolerated by the business. For example, a Customer Relationship Management (CRM) application or an Enterprise Resource Planning (ERP) application needs to be always available in a global enterprise, but if it is not available for 5 minutes of the day it does not cause a significant loss to the business. Similarly, email can be considered to be an enterprise critical application for similar reasons.

- **Critical**, which are desirable to be responsive and available but have greater levels of tolerance than mission and enterprise critical. For example, an information only site or a document management system or a library management system would come under this category.

Now let us discuss synchronous and asynchronous processing. Synchronous processing involves a user waiting for transaction completion, in order for the business processing to move forward. For example, while submitting an insurance claim, the end user waits for the IT system to respond in order to get a confirmation of acceptance of the claim. These are usually transactions that come under the realm of the front office of a business. Asynchronous processing on the other hand relates to back office processing, where a response is not immediately required. For example, once the claim submission is confirmed the end user can be given an acknowledgement number for further inquiry. The end user does not have to wait in front of a computer terminal until the claim is fully processed. Therefore the business processing of the claim is asynchronous with respect to the end user.

Finally, let us discuss workflow. In a workflow system transactions get generated within the system as a result of a business transaction initiated by an end user. For example, a customer wishes to withdraw a large amount of money from a bank. This withdrawal transaction results in a workflow for approval by the bank manager. As a second example, consider an employee appraisal system, where an employee fills up an annual self appraisal form. Upon submission of the self appraisal, the request is routed to the employee’s supervisor for appraisal, and this becomes part of a workflow.

During workload gathering, the use of the term “transaction” can be misleading for end customers. In a consulting engagement that the author had with an airline caterer, it took two weeks to realise that the customer’s interpretation of transaction was quite different from what we have meant in this document. To get around this communication gap, the customer was asked to present the organization structure in terms of department and types of users in each department. Then the customer was asked to list out the activities per user and explained that this is what we mean by “transaction.” Once this gap was bridged it took just one day to get a list of about 100 types of transactions across all departments.

Consider this example of a real life workload. One of the largest banks in the world processes close to 20 million banking transactions per day (including inquiries). If we wish to collect requirements as per the checklist given Table 1, we will realise
that there are 1200 types of transactions and it will not be possible in a short amount of time to collect data for each of transaction. Instead, we look at the most frequent transactions, samples of which are given in the table below and summarise the trend. In this case, it was observed that the top 10 transactions constitute 40% of the number of transactions per day, the top 50 constitute 80%, and the top 100 constitute 92% of the number of transactions per day.

Transaction mix for a very large bank

<table>
<thead>
<tr>
<th>Transaction Type</th>
<th>Percentage Occurrence</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cash Withdrawal (ATM)</td>
<td>6.5%</td>
</tr>
<tr>
<td>Cash Withdrawal</td>
<td>5.7%</td>
</tr>
<tr>
<td>Passbook Line – No Update</td>
<td>4.9%</td>
</tr>
<tr>
<td>Short Enquiry</td>
<td>4.8%</td>
</tr>
<tr>
<td>Update Passbook</td>
<td>4.7%</td>
</tr>
<tr>
<td>Cash Deposit</td>
<td>3.3%</td>
</tr>
<tr>
<td>Credit Transaction</td>
<td>2.6%</td>
</tr>
<tr>
<td>Deposit Transfer</td>
<td>2.2%</td>
</tr>
<tr>
<td>Cheque Deposit</td>
<td>1.9%</td>
</tr>
</tbody>
</table>

For the same bank, it was required to estimate the peak transaction workload. The first step involves examining the variations in number of transactions per day. It was observed that the on the day with maximum load across the year, the transactions are 30% more than average. After this step, one needs to examine the peak transactions per hour within the day. Data showed that 40% of transactions occur in peak period of 3 hours. The next step is to look at the intra-hour skew and narrow down to the peak requirement per minute or per second. For example, in a very large stock exchange 25% of the daily transactions occur in a window of 15 minutes.

1.2 Workload Gathering for Batch Processing

Batch processing occurs in bulk, either at a predefined period of the day or once in a given time period (for example, once a week or once a month). In this case there is no requirement of instantaneous response, but the requirement is that bulk processing complete in a given time period. For example, interest accrual across millions of account holders in a bank, cheque printing, data backup, employee salary deposit in a bank are all examples of batch processing. Table 2 provides a checklist for gathering workload inputs for batch processing.

<table>
<thead>
<tr>
<th>Table 2: Workload Inputs to be gathered for Batch Processing</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. List of business components that involve batch processing</td>
</tr>
<tr>
<td>2. For each business component, the batch processing workload:</td>
</tr>
<tr>
<td>a. List of batch programs</td>
</tr>
<tr>
<td>b. Mix of batch programs in terms of schedule and completion windows</td>
</tr>
<tr>
<td>c. Frequency of batch programs in terms of daily, weekly, monthly, and so on.</td>
</tr>
<tr>
<td>d. Complexity of batch programs in terms of number of records fetched and processed</td>
</tr>
<tr>
<td>e. The concurrency of batch processing with online processing (overlapping periods of time)</td>
</tr>
<tr>
<td>f. Schedule and duration of backup</td>
</tr>
</tbody>
</table>
1.3 Workload Gathering for Reports
Reports are required by various business departments as well as by management. Frequently occurring reports can be scheduled periodically, for example a daily summary of bank balances per branch of a large bank. On the other hand, there may be ad hoc reports that are requested whenever the necessity arises, as in the case of fraud analysis. Reports processing falls in between OLTP and batch processing. Table 3 provides a simple checklist for workload gathering for reports. It should be used over and above the checklists provided in Table 1 and Table 2.

Table 3 Workload Inputs to be Gathered for Reports

<table>
<thead>
<tr>
<th>No.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>List of all business components that involve reports</td>
</tr>
<tr>
<td>2.</td>
<td>For each business component, the reports processing workload:</td>
</tr>
<tr>
<td>a.</td>
<td>List of reports and classification as scheduled and ad hoc</td>
</tr>
<tr>
<td>b.</td>
<td>Mix of reports</td>
</tr>
<tr>
<td>c.</td>
<td>Frequency of reports as fine grained as possible</td>
</tr>
<tr>
<td>d.</td>
<td>Peak rate of each type of report</td>
</tr>
<tr>
<td>e.</td>
<td>Growth rate of reports</td>
</tr>
<tr>
<td>f.</td>
<td>The concurrency of report processing, batch processing, and online transaction processing</td>
</tr>
<tr>
<td>g.</td>
<td>Report complexity in terms of number of records processed and returned</td>
</tr>
</tbody>
</table>

1.4 Workload Gathering for Data Volumes
Business data growth impacts performance significantly, both through increased storage costs and the adverse effects of data volumes on the performance of transactions, reports, and batch programs. At the requirements gathering stage one may not have data sizes and volumes at the database level or the file level. However, it is necessary to have a business data model and capture the sizes and volumetric figures of business entities. For example, for an insurance business the entities would be customers, policies, claims, and the like. Table 4 provides a checklist for workload gathering around data volumes.

Table 4: Workload Inputs to be gathered for Data Volumes

<table>
<thead>
<tr>
<th>No.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Business data model across all business components</td>
</tr>
<tr>
<td>2.</td>
<td>The volumes of business and data entities:</td>
</tr>
<tr>
<td>a.</td>
<td>List of entities (for example, customers, orders, policies, claims)</td>
</tr>
<tr>
<td>b.</td>
<td>Data volume (number of records) per entity as of today</td>
</tr>
<tr>
<td>c.</td>
<td>Data access patterns, if any</td>
</tr>
<tr>
<td>d.</td>
<td>Growth rate of entities over one to five year period, preferably quarter on quarter growth</td>
</tr>
<tr>
<td>e.</td>
<td>Data retention period in number of years</td>
</tr>
</tbody>
</table>

Data access patterns are important to note because quite often it is observed that more than 80% of data accessed is by less than 20% of the transactions or on less than 20% of the data items. For example in a large brokerage 85% of the trades occur on less than 2% of the stocks. The last requirement on data retention period
is important from a compliance perspective. Some businesses need to keep data for 7 years, some for 25 years.

1.5 Workload Gathering for Users

As the business grows, so do the users of IT systems. Collecting data about users and their usage profiles not only helps one understand the business workloads better, but also provides a useful crosscheck against transaction and report workload data. Table 5 provides a checklist for workload gathering for users.

Table 5: Workload Inputs to be gathered for Users

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>List of business processing departments or groups</td>
</tr>
<tr>
<td>2.</td>
<td>For each department or group:</td>
</tr>
<tr>
<td></td>
<td>a. Classification of users (for example, e.g. business user, administrator, system manager, control personnel)</td>
</tr>
<tr>
<td></td>
<td>b. Number of registered users per type of user</td>
</tr>
<tr>
<td></td>
<td>c. Number of concurrent users per type of user, as a function of time of day</td>
</tr>
<tr>
<td></td>
<td>d. The types of transactions that can be executed by each type of user</td>
</tr>
<tr>
<td></td>
<td>e. Growth rate of users, on an annual basis</td>
</tr>
<tr>
<td></td>
<td>f. The interactions between user and end customer, if any (for example, questions asked in telephone call)</td>
</tr>
</tbody>
</table>

This step may seem redundant to some, but it is a useful data gathering processing in order to validate OLTP or reporting workload.

One of the most misinterpreted terms in workload gathering is the term ‘concurrent users’. Quite often people misinterpret it to be number of users hitting the enter key at the same time, or number of ‘concurrent’ web server hits. We should note that since we are gathering workload at the business level, it is most appropriate to define concurrency in terms of number of business users who are working on the IT system (more technically number of sessions with the web servers that have not timed out). Thus there may be 3000 concurrent users, and some could be doing nothing at any given point in time, but that can be factored as ‘idle time’ or ‘think time’ and will automatically reduce the transaction workload on the system.

1.6 Workload Gathering for External Interfaces

With expansion in many businesses, the workload does not emanate from business users alone. The business usually creates channels for the external world to interact with it, be it customers, or partners, or suppliers. For example, in the banking world ATMs have become very common and all ATM transactions come to a bank’s IT systems via interface channels. The interface requests could be just like OLTP or if processed periodically they could be processed in batch mode. Table 6 provides a checklist for gathering workload inputs for external interfaces.
Table 6: Workload Inputs to be gathered for External Interfaces

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>List of all business components that involve interfaces</td>
</tr>
<tr>
<td>2.</td>
<td>For each business component:</td>
</tr>
<tr>
<td></td>
<td>a. List of interfaces that carry transactional workload</td>
</tr>
<tr>
<td></td>
<td>b. List of interfaces that carry batch workload</td>
</tr>
<tr>
<td></td>
<td>c. For transactional interfaces collect data as per Table 1</td>
</tr>
<tr>
<td></td>
<td>d. For batch interfaces collect data as per Table 2</td>
</tr>
</tbody>
</table>

1.7 Some Tips on Workload Data Collection

Collection of workload metrics is not always straightforward in most businesses. Acquiring a list of transactions, reports, and batch jobs is normally easy, but getting a fine grained rate of transactions per minute or per second is difficult. For a system in production this data can be obtained by means of analysing transaction history data in databases or system logs that contain timestamps. For a new system that has yet to be implemented one has to make several assumptions on transaction mix, peak to average traffic, as well as peak usage hours.

Data volumes are easier to estimate if they are directly related to the business, and for an existing system they can easily be captured by analysing counts from database tables. Information about the number of registered users is usually available but estimating the number of concurrent users is never straightforward. For a new system one can relate concurrent users and transaction workload. For an existing system it is best to measure concurrent users by means of monitoring network connections and analysing web and system logs. Data returned over the network can be measured using network monitoring tools or web server logs. Early on, it is also useful to set targets for this metric so that network consumption can be well balanced.

1.8 Gathering Inputs on Performance Targets

In addition to collecting inputs on workloads, the performance requirements gathering phase also involves collecting inputs on performance targets. The performance metrics for a system typically are:

1. **Response time per online interaction**: This is the time taken to get a response for a screen interaction. Response time is usually measured as an average value or a percentile value. For example, for 95th percentile less than 2 seconds, means that 95 per cent of the interactions must complete in less than 2 seconds. These are some factors to be considered while setting response time targets:
   a. Based on type of interaction, for example, a simple hyperlink click may have a smaller target than a transaction submit.
   b. Based on network bandwidth, for example, intranet response time of 2 seconds and internet response time of 10 seconds.
   c. Based on overall completion time of transaction – see the next point.

2. **Completion time per online transaction**: This is the time to navigate all screens to submit and complete an online transaction. This is inclusive of
data entry time, response time, time to browse the result of a screen interaction, and time to select next screen within part of the transaction, time to collect information from end customer (over telephone), if relevant. This can play an important role in setting of response times. For example, if a user is to complete a transaction in 5 minutes, and the transaction has 10 web interactions to complete, where each transaction requires 20 seconds to fill up and review, the average response time per interaction should be no more than \((5\times 60 - 10\times 20)/10 = 10\) seconds.

3. **Delivery time for asynchronous transaction**: This is important in email and workflow environments. The response time for an email operation is the time required to submit an email to the email server, but what is important to the receiver is the time by which the email will reach the receiver, which is the delivery time. These are some factors to be considered when setting completion time targets:
   a. Based on type of transaction
   b. Based on network bandwidth
   c. Based on number of stages or hops in application/system architecture before reaching final destination

4. **Transaction throughput**: While response time is a customer metric, throughput is a business metric. Throughput refers to transactions processed per unit time. For example, Bank XYZ needs to process 1 million deposit transactions per day, at a peak of 500 transactions per second between 1 PM and 2 PM every Monday. The throughput can be classified by peak and average, but it is important to be as fine grained as possible in terms of time. If throughput is given in transactions per year, the peak per second is not easy to estimate without making a list of assumptions that may not be realistic. Throughput also needs to be classified per type of transaction.

5. **Batch completion time**: The completion time needs to be specified by type of batch program, including backups. For example, daily interest calculation can start any time after 12 AM and should complete before 6 AM. The batch completion time should also be specified relevant to the time of year, for example, day end, weekend, month end, and year end, since each of those may have different processing loads.

6. **Report throughput and completion time**: This has to be classified into scheduled and ad hoc reports. Scheduled reports can be treated as being similar to batch operations, and very simple reports can be treated similar to transaction processing. Per type of report, throughput should be stated for average and peak workloads. For ad hoc reports, the completion time may be stated but it is required to have a discipline in place to control the submission of ad hoc reports. Ad-hoc reporting is better done by having a separate reporting system or a data warehouse in place.

7. **Resource consumption**: At the operating system level, resource consumption drills down to just four metrics:
a. **CPU utilization**: This is the percentage of time the CPUs of the system are busy. An upper bound can be specified so that capacity planning becomes an easier process. On most open systems, it is desired to have CPU utilization no more than 70% for online transaction processing workload.

b. **Memory consumption**: This is the number of MB or GB consumed of the system’s RAM. Typically, the online and batch processing are measured separately.

c. **Disk consumption**: Disk consumption is measured by amount of GB consumed by the application and needs to be reviewed quarter on quarter. Disk utilization and I/Os per second of the disks subsystem are often measured to plan for capacity more holistically.

d. **Network bandwidth**: In terms of Kbps or Mbps, it is useful to set a target of network consumption overall as well as per user. For example, 20 Kbps per user and 2 Mbps overall from a branch to a centralized server. Network bandwidth targets are sensitive to application and technical architecture, apart from business processing workload.

2. **Qualitative Requirements Analysis**

This section first covers ten real life examples of how performance can be managed effectively during requirements analysis itself. Thereafter, a more formal approach to qualitative performance requirements analysis is proposed.

2.1 **Ten Real Life Examples**

**Example 1: Selecting one from many**

A securities depository participant offers a call centre service to its customers. The normal type of inquiry of a customer is to know about security holding details, and transaction details. The customer provides his **InvestorId** which should be entered by the call centre operator, after which all details of the customers holdings and transactions in specific securities can be browsed through. A typical depository participant has 10,000 to 100,000 customers.

While an input field is provided to type in a 6 digit **InvestorId**, there is also an option to click a drop down box that will show all the **InvestorIds**. The operator, more often than not, clicks on the drop down box to get the **InvestorIds** in sorted order, and browses down the list until the required **InvestorId** is found.

The drop-down box option is the preferred option for the operator; however, consider the wastage of resources and its impact on performance. Instead of typing in a 6 digit number, the system has to retrieve as many as 100,000 **InvestorIds** in

---

1 The securities depository has about 100 depository participants as a network of branches.
the worst case. This not only consumes significant CPU cycles, memory, and disk I/Os but also means 600KB of data traversal on the network to the front end.

Instead, typing the 6 digit investor will bring down resource consumption significantly and offer quick response from the system.

Example 2: Inquiries from the hierarchy

A stock exchange caters to 10,000 traders, several of whom are not independent traders. In the typical case, several traders operate from a branch, whose management is interested in inquiring about its traders’ activities. The total number of trades per day across all trades is close to 1 million, and the total number of inquiries from managers close to 10,000 per day.

Every order placed by a trader has to update an activity database, which has 4 indexes for regular inquiries by traders. To satisfy manager inquiries, another 4 indexes had to be added. These indexes made the inquiries fast but slowed down the order processing, since every order had to update 8 indexes instead of 4.

The manager inquiry feature turned out to be a disaster for the stock exchange since 99% of the transactions were slowing down to speed up the remaining 1%. This left the stock exchange with no option but to disable the manager inquiries online and provide them with day end reports instead. While this was done as a stop gap measure, this change was mandated due to its effectiveness and has been in place for the last decade years at the exchange.

Example 3: Pay per use and charge back

The computer sciences department of a certain US university provides laser printer access to all students and faculty. Students in courses involving programming often take print-outs of their code, which runs in to a large number of pages. Not only is this costly for the department but it also causes a significant wait time for faculty and other students who need to collect their one or two page print-outs. To effectively manage this situation the department implemented a quota system for students. The first 200 pages were free, after which there was a charge of 10 cents per page. As expected, most students refrained from printing out huge programs on laser printers.

A large financial services company has users who access SAS. Several users make ad hoc queries on the mainframe some of which take hours. In cases where these queries are running for long periods of time, the user does not bother to abort his query and it is left to the system administrators to notice large consuming jobs on the mainframe and take disciplinary action. The problem is compounded by the fact that the company has to pay the mainframe vendor in proportion to the CPU cycles consumed per month. To enforce a discipline, the company decided to implement a charge back policy, wherein every CPU cycle consumed was charged back to the project using it.
Example 4: Scheduling of use

An online employee appraisal system went live for the first time in the history of a software services company. The load on the 2 CPU server on day one was so high that the system crashed. As a result, the application was hosted on a 4 CPU server. This gave some relief but it was noticed that after the first few days of going online, the server was idle most of the time. More than 25% of the appraisals were complete on the day of the deadline. This got the human resources department into action, and a plan was laid down such every branch of the company would complete its appraisals on specific days in a controlled manner.

Example 5: Timer driven broadcast

A certain stock exchange provides an update of stock prices to traders by means of a broadcast. Whenever a stock price changed the system broadcast the change, and this implementation was called event driven broadcast. In a short span of time the volumes grew 20 fold and during peak periods there were hundreds to thousands of trades per second. The event driven broadcast consumed significant CPU time as well network bandwidth. In fact the network bandwidth choked to the point that packets were dropped and traders could not get an accurate picture of the market. Moreover, even if the packets had been delivered the rate would be too fast for the trader to take notice of the events.

Upon analysis of the 1400 stocks it was found that the top 5 stocks account for 50% of the trades, and the top 30 stocks account for 90% of the trades. This meant that most of the time the trader was receiving updates on the same stock several times a second. This led to a change in broadcast requirement, which was called timer driven broadcast.

Under the timer driven broadcast, the top 30 stocks had a configurable timer parameter, which was set to 2 seconds. The cold stocks had their parameter set to 10 seconds. If a trader wanted to see any cold stock faster than the refresh rate, he could always inquire on its latest price and receive a response within 2 seconds. This was not only satisfactory to the trader but it also reduced resource consumption very significantly, and has been running successfully for the last 10 years at the exchange. [Note that this example applies only to human traders and does not apply to algorithmic trading.]
Example 6: First summary, then details

A large organization implemented an employee portal, which provided a search facility for a given employee. A registered user of the portal could inquire about details of an employee by providing the employee's name or a subset of the name. The portal would return all matching entries with all details about the employee such as name, current address, telephone number, department, project, supervisor, as well as details of previous projects.

Quite often the matching entries were too many in number. For example, if someone typed “John”, it would result in a display of hundreds of employee records. While specific attributes such as last name and department were useful to narrow down to the right “John”, the rest of the fields displayed were redundant for the other “Johns.” This not only cluttered up the browser, but also resulted in a large amount of data traversing the network. It was proposed to provide a table of contents format instead, so that after narrowing down to the right employee, his details could be obtained by clicking on the link for that employee.

Example 7: Flow control

A certain stock exchange provides access to individual traders as well as to brokerage houses. Brokerage houses host a large number of traders and the orders emanating from them often occur in bursts for the stock exchange. Quite often these bursts cause a denial of service to individual traders since all orders on a given stock must be treated on a first come first serve basis. To better manage the system performance, a flow control feature was introduced by the exchange for orders arriving in bursts from brokerage houses. Two parameters were used for flow control: the number of orders per burst, and the delay per burst. This regulated the flow of orders to the exchange and provided better service to individual traders.
Example 8: Avoid being fancy

The last mile problem is a well-known issue in the Internet. There continue to be many users who dial up using 56 Kbps modems. Certain sites such as www.google.com ensure that their web pages are small – with a home page of 3 KB and results from a complex search coming to less than 20 KB. On the other hand, certain online retail companies prefer to use several fancy features on their home page. In a specific case, the home page of an online retail site was close to 100 KB. Anybody accessing this site on a dialup connection would have to wait for close to half a minute just to see the home page displayed.

One may argue that with decreasing bandwidth costs, this point may not be applicable. One should note that bandwidth is not just the only factor, round trip times matter too. If your web page necessitates too many round trips between the browser and the web server, then no matter what be the bandwidth, a user in another country or continent will not be able to get acceptable response times (unless you replicate the content on a local server in that country).

Example 9: Differential response for today and for yesteryears

In a specific securities depository, customers often call up for their statement of transactions. Often the requirement is for the current quarter. However, the operators prefer to use a wild card search option to display all transactions. The system also keeps dumping data day-by-day on to the same set of tables, which causes degradation over time. Experiments on retention of only 3 months data in primary tables showed a 10 fold improvement in performance. As a result, the depository went for a reporting and archival solution, where differential response times were guaranteed depending on the age of the data.

Example 10: Limits on business requirements

One of the commodities exchanges required a risk management system that initially required more than 1000 CPUs. One of the easiest ways to reduce the capacity was to introduce a limit on the maximum number of defaulters. Once this number was known, the architecture and design analysis could be optimized to meet the processing requirements for the given limit. This resulted in the number of CPUs being reduced by over a factor of 100.

2.2 Methodology for Qualitative Performance Requirements Analysis

Based on the examples described above, a methodology is provided in this section for analysing performance requirements. We also supplement it with a performance requirements analysis matrix to enable an analyst to quickly arrive at a recommendation for better performance management.

The qualitative performance requirements analysis methodology is provided in Table 7.
Table 7: Methodology for Qualitative Performance Requirements Analysis

1. Conduct a thorough workload analysis, as described in Section 1
2. Set performance targets as discussed in Section 1.
3. Review performance targets in light of technology being used such as dialup lines, low end PCs as desktops, etc.
4. Analyse transactions/interactions that result in a redundant display of information (such as those in Examples 1, 5, and 6 in Section 2.1). Evaluate filtering alternatives for the same.
5. Analyse transactions, reports, batch jobs that can cause excessive consumption of system resources. For these evaluate options for consumption reduction:
   a. Pay per use or charge back (as in Example 3)
   b. Reducing fancy features (as in Example 8)
   c. Creating categories and providing better performance to those with higher priority (as in Example 9)
   d. Creating requirements that will limit resource consumption (as in Example 10)
6. Evaluate transactions, reports, batch jobs that require workload management, and arrive at a suitable workload management policy such as:
   a. Flow control (as in Example 4)
   b. Scheduling (as in Example 7)
7. Analyse transactions, reports, and batch jobs that can potentially interfere with each other’s response. Transactions that are less frequent but can cause significant degradation to very frequent transactions are candidates that require a change in business processing (as in Example 2).
8. Evaluate adding requirements for tracking performance of specific classes of transactions or for all types of transactions, so that it is easy to resolve performance issues in production, in particular, when there are multiple vendors.

In the above methodology, Steps 4 through 7 require the following factors for analysis of transactions:

- Transaction frequency
- Transaction complexity in terms of resource consumption estimates
- Transaction workload in terms of bursty or streamlined
- Transaction interference (Step 7)

If we qualify frequency and complexity as low or high, workload as bursty or streamlined, and interference as yes or no, we can arrive at the Performance Requirements Analysis Matrix shown in Table 8.
Table 8: Performance Requirements Analysis Matrix

(To be analysed for each business transaction, report, batch job)

<table>
<thead>
<tr>
<th>Frequency</th>
<th>Complexity</th>
<th>Workload</th>
<th>Interference</th>
<th>Recommendation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Low</td>
<td>Low</td>
<td>-</td>
<td>Yes</td>
<td>Evaluate options for change in business processing</td>
</tr>
<tr>
<td>Low</td>
<td>High</td>
<td>-</td>
<td>Yes</td>
<td>Evaluate consumption reduction options such as pay per use, charge back, limits to consumption, prioritization</td>
</tr>
<tr>
<td>High</td>
<td>Low</td>
<td>Bursty</td>
<td>-</td>
<td>Evaluate workload management such as flow control, scheduling</td>
</tr>
<tr>
<td>High</td>
<td>High</td>
<td>Streamlined</td>
<td>-</td>
<td>Evaluate consumption reduction options</td>
</tr>
<tr>
<td>High</td>
<td>High</td>
<td>Bursty</td>
<td>-</td>
<td>Evaluate consumption reduction and workload management options</td>
</tr>
</tbody>
</table>

To execute a thorough requirements analysis study for performance it is necessary to have a team with the following composition:
- Business analysts
- Application architects
- Performance engineers/Technical Architects
- System administrators (for analysing systems already in production)

3. Quantitative Performance Requirements Analysis

This section utilizes Little’s Law (described below) to analyse performance requirements in three scenarios:
- Deriving performance targets
- Validating performance targets
- Analysing interference across multiple types of transactions because of targets specified
3.1 Little’s Law

Little’s Law is a well known theorem from operations research, which was published in 1961 by J.D. Little. We use a simplified version of it for closed systems (having a fixed number of users). Consider a system, as shown in Figure 2, with \( N \) (business) users, with an average response time of \( R \) seconds per business transaction, and having a throughput of \( X \) business transactions per second. (We could might as well as use business interaction or web page per request.) Let us assume that the average think time per user is \( Z \) seconds, per business transaction. By think time we mean the time that a user spends in data entry, or in viewing results, or in doing anything other than waiting for a response.

![Figure 2: Closed System](image)

Now, as shown in Figure 2, the average cycle time for one user is \( C = R + Z \) and therefore the throughput per user is \( 1/(R+Z) \). For the overall system we get:

\[
\text{Little’s Law for Closed Systems: } R = X \cdot C = X (R + Z)
\]

The only assumption behind this law is that the system is work conserving, in that no work is generated within the system and no work is destroyed by the system. Now let us see some interesting applications of Little’s Law.

3.2 Deriving Performance Targets

Performance targets that are derived due to relationships between transactions, batch jobs, and reports can be estimated based on business workload models. However, very often customers provide number of concurrent users and response time targets, but not throughput targets. Or at times the throughput targets and number of concurrent users are given but not cycle time targets. Or at times throughput and cycle time targets are given but not number of concurrent users.

Using Little’s Law we can derive the missing target. To recapitulate, Little’s Law states that for any work conserving system (with end users included as part of the system):
where \( N \) is the (average) number of concurrent users, \( X \) is the system throughput, and \( C \) is the average system cycle time, which is the sum of average response time \( R \) and average think time \( Z \). Thus for example, if a system has 3,000 concurrent users and a throughput target of 100 transactions per second, the average cycle time per transaction is \( 3,000 / 100 = 30 \) seconds. If the think time per transaction is 25 seconds, then the average system response time target is \( 30 - 25 = 5 \) seconds.

### 3.3 Validating Performance Targets

As discussed above quite often customers provide two out of the three terms in Little’s Law. During a request for proposal (RFP) one often gets the chance to ask for clarifications from the customer and it is advisable to ask for the third target and validate using Little’s Law.

A real life case study will illustrate this point. The income tax department of a large country wanted to centralize its annual returns processing. This was a big risk that had to be mitigated since the tax department wanted to ensure that centralization would work. As result, the income tax department floated an RFP asking vendors to evaluate the performance of the centralized architecture and after achieving an acceptable performance propose the most cost effective hardware solution for the system.

The performance had to be evaluated in a benchmark centre in a hardware vendor’s laboratory. The RFP stated a requirement of 4,000 assessing officers and an average cycle time of 10 seconds for a single assessment. The cycle time included entering the tax payer’s account number, reviewing the written tax submission versus what showed up on the system’s screen, then deciding whether to refund tax or not, and finally saving and printing the output before moving on to the next return.

The 10 second target seemed far too aggressive and the customer was not ready to relax it. Refusing to participate in the bid meant loss of a very large deal for the company. Therefore, taking this further, the customer was asked to state the throughput target (note we have number in the system and cycle time, so what is missing is the throughput target in order to apply Little’s Law). During the RFP process, it was clarified that the target was 52 million assessments per year.

Now, applying Little’s Law to these targets:

- \( N = 4,000 \)
- \( C = 10 \) seconds
- \( X = 52 \) million per year

Applying Little’s Law we get:

\[
X = N / C = 4,000 / 10 = 400 \text{ returns/sec}
\]

Assuming a 5 hour working day we get:
\[ X = 400 \times 3,600 \times 5 = 7.2 \text{ million assessments per day} \]

The result of this essentially means that the annual target of 52 million assessments would be met within 8 working days!

After this was presented to the income tax department, they clearly realized the aggressive nature of the RFP and asked for an assessment of the cycle time. For this purpose, we assumed that a peak to average workload ratio of 3:1. In other words, complete 52 million returns in 4 months rather than 12 months. Assuming 20 working days per month and 5 working hours per day, we get:

\[ X = \frac{52,000,000}{4 \times 20 \times 5 \times 3,600} = 36 \text{ returns/ sec} \]

Now apply Little's Law to get:

\[ C = \frac{N}{X} = \frac{4,000}{36} = 111 \text{ seconds} \]

We thus proposed a cycle time of 2 minutes for the tax processing system.

Initially the tax department did not agree to this target and revised the cycle time to 45 seconds from the original target of 10 seconds. After weeks of benchmarking, the revised target of 45 seconds was also not good enough for any of the vendors to meet the performance targets. However, when benchmarked with a cycle time of 2 minutes, the response times were quite acceptable. This resulted in the original RFP being scrapped and a new RFP was released with cycle time of 2 minutes!

Note the power of Little’s Law. Without the need to run a single test, without the need anything about the technology used, without the need to know about the wide area network impact on centralization, without the need to know the impact of processing and storing millions of tax returns, we could arrive at the right requirements and get a complete RFP rewritten (thus ensuring sound parameters were considered in this solution for a critical national requirement). The power of Little’s Law comes out very well here, with one simple relationship between number of users, throughput, and response time. For another interesting example see Exercise 5 at the end of this document.

3.4 Analysing Interference between Transactions or Different Workload Types

One of the biggest challenges in IT systems is that a system that is proven for one type of transaction often starts stuttering when additional business transactions are added. Often a system supposed to work for OLTP has to process a significant amount of reporting workload. To compound the problem, reports are often ad hoc. At that stage it becomes difficult to predict interference of reporting workload on OLTP, but a good analogy is that the situation is similar to running a virus scan on your PC or laptop.

It would be desirable to quantify the impact of reporting workload on OLTP at requirements analysis itself, rather than have to wait for a disaster to occur in production. So how does one analyse such an impact during requirement analysis?
To proceed with the analysis we first need to take Little’s Law further. Consider a system with two types of workload (or two types of transactions) as shown in Figure 3.

**Figure 3: Analysis for Multiple Workload Types**

In Figure 3, two types of traffic are shown. A pink stream constituting OLTP transactions and a blue stream constituting reporting requests. The question we would like to pose is:

**Does Little’s Law hold for each stream?**

In other words, consider that you are given pink glasses through which you can only see the pink stream (that is, OLTP) and that you cannot see the blue stream. Or consider that you are now wearing blue glasses through which you only see the blue stream (that is, reports) and that you cannot see the pink stream. The question is, will Little’s Law hold in each case, despite the fact that in reality there are two streams running through the system (even though you may see only one at a time)?

The answer is yes. If we go back to the derivation of Little’s Law, we will see that it has been proven from an observer’s point of view and not necessarily from a system point of view. As long as the traffic that the observer sees is work conserving Little’s Law will hold true for the observer.

So now let us go back to our problem of interference caused on OLTP transactions because of reporting workload. Take the following example to illustrate this problem. We are given the following requirements for OLTP and reports that will share a common database server:

- Number of OLTP users: 1000
- Average cycle time of OLTP transactions: 10 seconds
- Number of reporting users: 50
- Average cycle time of reporting requests: 25 seconds

In addition, we also have some design level information at the database:

- Average number of database requests per OLTP transaction: 2
- Average number of database requests per report: 50
Now let us apply Little’s Law to each stream.

- Throughput of OLTP transactions = 1,000 / 10 = 100 transactions per second
- Throughput of Reports = 50 / 25 = 2 reports per second

At this level it may appear that the reporting load is too small to impact online transactions. However, remember the interference is at the common database server. So we need to multiply by the visit counts at the database to get:

- OLTP throughput at database = 100 x 2 = 200 requests / second
- Reports throughput at database = 2 x 50 = 100 requests / second

We can now see that there is a substantial load on the database because of reports. It is typical for reports to have more complex database queries than OLTP, which means that reports processing at the database will take up at least 100 / (200 + 100) = 33 % of the database capacity.

Note that we could measure this impact solely on the basis of requirements, without the need to test the software or take any measurements. Thus Little’s Law provides a very powerful technique to analyse requirements impact right at the requirements stage itself.

4. Concluding Remarks

This chapter has examined a very important phase of performance engineering that is usually skipped in the traditional software development and maintenance lifecycle. The power of performance requirements analysis has been illustrated through a series of examples, both qualitatively and quantitatively. Proper use of Little’s Law not only helps one derive performance requirements correctly but also helps validate requirements upfront as well as determine impact of conflicting requirements.

The whole philosophy of performance requirements analysis centres around one important theme: avoid work if it is not required. The main question to ask during analysis is why there is such a requirement in the first place and is the processing demanded really required for that purpose. If you can do this right, then you can save many person years of effort in avoiding implementation as opposed to engineering the implementation. A nice joke in this context will illustrate what we intend to convey.

There were three kids who were fighting on whose father is the fastest in the world. The first kid was from Japan, the second from America, and the third from India. The Japanese kid confidently said, “My dad is the fastest.” When asked to prove it he said that if you take a bow and arrow and shoot at a target, his dad could race the arrow even before it could hit the target. The American kid retorted by saying, “My dad is faster.” When asked to prove it he said that if you shoot at the same target with a handgun, his dad could race the bullet even before it could hit the target. Then it was the Indian kid’s turn. While he was sure his father was nowhere as fast as a bullet, he had to protect the image of himself, his father and his country. So the Indian kid said, “My dad is the fastest.” This surprised the other two kids and they vehemently asked him to prove it. To which he said, “My dad
works in a government office. His office closes at 6 o’clock but he comes home by 4 o’clock!”

This goes to show that performance is not just about engineering; it just as much about avoiding things that should not be done in the first place. This is where performance requirements analysis provides tremendous benefits to organizations.

**Useful Reading**

The material presented in this document is derived in part from some of the references below. Other references have been added as useful reading to supplement what is presented in this document.


**Exercises**

1. A large airline catering company needs to plan on capacity of its IT systems. You are called in as the consultant for this exercise. You meet the IT manager and as per the checklists provided in Section 1, you gather performance requirements. However, whenever you ask the IT manager for types of transactions you get the answer that there is only one type of transaction. You are convinced that such a large organization will have many types of transactions. What should you ask the IT manager, to make progress in your requirements gathering exercise:
   a. The CPU utilization of the current hardware
   b. The number of requests to the database server
   c. The number of requests to the application server
   d. What is the organization structure, what are the different departments in company, what activities do the users do in each department, what interactions do the users have with external entities such as airlines, and what reports are run for the management on a daily basis?

2. With the advent of Internet-based trading, stock exchanges worldwide are faced with a change in workload pattern. At brokerage firms, orders are
accumulated from Internet traders post trading hours. The accumulated orders are then pumped in to stock exchanges at the start of the next trading day. You are the CIO of a large stock exchange and you need to manage this surge in traffic at the start of the day. Based on your learning from this chapter, what approach would you take, that would be accepted by the management?

a. Get a performance engineer to optimise the stock exchange software, so that it can handle the additional load
b. Ask the management for an increase in budget to meet the additional load
c. Put a throttle on the order pumping rate per brokerage. If the pumping rate on a given connection exceeds the limit specified, drop the connection. This will force brokerage firms to buy more connections from the exchange and in turn you can utilize the money to add capacity.
d. Issue a letter to all brokerage firms that they need to disallow orders post trading hours

3. A large government ministry has outsourced a large project to you. The project involves building a system that will allow for company returns and balance sheets to be uploaded to the ministry’s portal. The investment in hardware and software is to be borne by you. In return you can collect money from each company as your processing fee, provided you give them a satisfactory level of response. Within a short time you realise that there are many small companies which upload documents with less than 5 pages, and there are few very large companies which upload documents with more than 100 pages. Whenever a large company is uploading its documents, uploads made by other companies tend to get slowed down (like following a truck on a highway). What alternatives would be useful for you to try out (more than one answer can be correct here)?

a. Keep your processing fee proportional to the number of pages per document
b. Forgo your processing fee during intervals when large companies upload their documents
c. Provide an upload tool that slows down the upload for large documents, whenever there are concurrent uploads in progress
d. Allow electronic submission only for small sized companies

4. An insurance company has a cycle time of 15 minutes to issue a policy to a customer. The company intends to cater to a demand of 12,000 policies per hour. How many concurrent users could be supported, for issuing policies?

5. Program trading has changed the financial services world due to its demand on low latency and high throughput from a small number of program trading terminals. You are in charge of a program trading implementation that needs to cater to 10 terminals, 1000 orders/sec, and an average order cycle time of 100 ms. Under normal circumstances you would take this requirement and worry about it only when you have your software ready for performance testing. At that stage it will be very late in the cycle if you realise that there is a flaw in the requirements, and you would rather have
this debate raised early on, during the requirements analysis cycle. Is the specified requirement achievable?

6. You are called as a consultant to a call centre, which is targeting 30,000 calls per day. 75% of the calls occur during a peak window of 3 hours. The average call duration is 5 minutes. The call centre management wishes to keep employee utilization at 70% during peak hours. How many employees does the call centre need to manage this workload?

Answers to the exercises:

1. d
2. c
3. a, c
4. By Little’s Law \( N = \frac{12,000}{60 \times 15} = 3,000 \) concurrent users
5. By Little’s Law \( N = X \cdot C = 1,000 \times 0.1 = 100 \), but \( N \) is given as 10, so this is impossible. Even if you try to optimise the cycle time you need to get it down to 10ms for the requirement to be feasible. But network latencies from brokerage firms to stock exchanges are much higher. In short, this is an infeasible requirement that needs to be highlighted at the start of the project.
6. Throughput \( X = 0.75 \times 30,000 / 3 = 7,500 \) calls/hour = 125/minute. Average cycle time = average call time / employee utilization = 5 / 0.7 = 7.1 minutes Average number of employees required = \( X \cdot C = 125 \times 7.1 = 888 \)