Best Practices for Writing and Managing Performance Requirements: A Tutorial

André B. Bondi
Siemens Corporation, Corporate
Research and Technologies
755 College Road East
Princeton, NJ 08540 USA
+1 609 734 3578
andre.bondi@siemens.com

ABSTRACT
Performance requirements are one of the main drivers of architectural decisions. Because many performance problems have their roots in architectural decisions, and since poor performance is a principal cause of software project risk, it is essential that performance requirements be developed early in the software lifecycle, and that they be clearly formulated. In this tutorial, we shall look at criteria for high-quality performance requirements, including algebraic consistency, measurability, testability, and linkage to business and engineering needs. While focus of this tutorial is on practice, we shall show how the drafting of performance requirements can be aided by performance modeling. We shall show methods for presenting and managing performance requirements that will improve their chances of being accepted by architects, developers, testers, contract negotiators, and purchasers; and of their being successfully implemented and tested.

Categories and Subject Descriptors
C.4 **[Computer Systems Organization]**: Performance of systems; D.2.1 **[Software Engineering]**: Requirements specification.

General Terms
Measurement, documentation, performance.

Keywords
System performance, performance requirements.

1. INTRODUCTION
Poor computer system performance has been called the single most frequent cause of the failure of software projects [13, 1]. Among the causes of poor performance are poor architectural choices and inadequately specified performance requirements. In our experience and that of several performance engineers and developers with whom we have spoken, performance requirements may be vaguely written, or might not even have been written at all by the time the project is close to completion.

The absence of performance requirements increases the risk that performance will receive inadequate attention during the architectural, development, and functional testing phases of a software project. Performance requirements are drivers of computer and software architecture.

Since performance problems often have their roots in poor architectural decisions, the early establishment of performance requirements for a new system is crucial to the project's success.

The lack of well specified performance requirements places the burden on the performance tester to identify the ranges of workloads to which the system should be subjected before delivery. In such cases, performance testers must make conjectures about the anticipated system load and use case mix, and then devise load tests accordingly. If the tests are well structured, they can tell us whether the system has desirable performance properties under designated loads, but the tested loads may not be close to those envisioned for the system.

We have found that poorly written performance requirements incur an insidious cost. They cause confusion among the developers charged with meeting them, as well as among the performance testers who must verify that they are met. The confusion must be resolved in meetings to try to understand what was meant. In the author's experience, clarification of the performance requirements often means rewriting them in keeping with the spirit in which they were meant, and then communicating the revisions to the various stakeholders for approval.

Whether or not they are well formulated, performance requirements are a key ingredient of customer expectations of what the system will do. Therefore, they may constitute part of an agreement about what the supplier is supposed to deliver. It follows that poorly drafted requirements increase the prospect of incurring customer ill will, which can have undesirable consequences, including loss of business and even litigation.

In this tutorial, we identify best practices for specifying performance requirements, and examine the risks and pitfalls of building a software system in cases in which the requirements are either absent or written in a form that makes them inherently untestable or unachievable. We argue that performance requirements should possess the same good qualities as functional requirements, such as traceability, measurability, and unambiguousness [6], and be linked to business, engineering, or regulatory needs. By ensuring measurability, we reduce the possibility of adopting performance requirements that are unachievable or that impose constraints that have no impact on performance at all. The linkage may help to overcome the problem that slogans masquerading as performance requirements are sometimes neither achievable nor testable. Demanding this

ICPE'12, April 22–25, 2012, Boston, Massachusetts, USA.

linkage also ensures the traceability of a performance requirement by answering why the requirement was specified in the first place.

Confusion can also arise when a performance requirement has been written for no apparent reason, or when it is in conflict with other performance requirements. The guidelines we propose are intended to reduce the risk of these difficulties.

2. RELATION TO PREVIOUS WORK

Smith and Williams state that performance requirements are a precondition for good system performance [13]. Nixon describes a framework for managing them during the course of a project, and how they should be arranged to facilitate the identification of performance goals [10]. Ho *et al* [5] advocate the incremental formulation of performance requirements and conduct of performance tests in an Agile development process. They identify three levels of performance requirements. Level 1 performance requirements are related to system performance in single user mode. Level 2 performance requirements specify the performance requirements when the system is supporting a specific transaction rate. Level 3 requirements are similar to Level 2 requirements, but relate to the peak rate.

The emphasis in the present tutorial is on the practice of performance requirements engineering. We begin with a discussion of the performance metrics that are used in performance requirements, because a clear definition of performance metrics is necessary to ensure that performance requirements are unambiguous. As we describe below, the desirable attributes of performance requirements are closely related to those of functional requirements, for example as described in [6].

3. PERFORMANCE METRICS

It is essential to describe the performance of a system in terms that are commonly understood and unambiguously defined and related to the problem domain. Performance is described in terms of quantities known as *metrics*. A metric is defined as a standard of measurement [14]. Metrics should be defined in terms that aid the understanding of the system from both engineering and business perspectives. A comprehensive discussion of performance metrics is contained in [7]. The values of performance metrics should be obtainable by direct measurement or by arithmetic manipulation of direct measurement or the values of other metrics. Performance metrics such as average response times are based on sample statistics. Performance metrics such as average utilizations are based on time-averaged statistics.

A performance metric should inform us about the behavior of the system in the context of its domain of application and/or in the context of its resource usage. What is informative depends on the point of view of a particular stakeholder as well as on the domain itself. For example, an accountant may be interested in the monthly transaction volume of a system. By contrast, an individual user of the system may only be interested in its response time during a period of peak usage. This means that the system must be engineered to handle a set number of transactions per second in the peak hour. The latter quantity is of interest to the performance engineer. It is only of interest to the accountant to the extent that it is directly related to the total monthly volume. If the two metrics are to be used interchangeably, there must be a well known understanding and agreement about the relationship between them. Such an example exists in telephony. In the latter part of the 20th Century, it was understood that about 10% of the calls on a weekday occur during the busy hour. In the USA, this is true of both local call traffic and long distance traffic observed concurrently in multiple time zones. It is also understood that the number of relevant business days in a month is about 22. Thus, thus the monthly traffic volume would be approximately 22 times the busy hour volume, divided by 10%. For example, if 50,000 calls occur in a network in the busy hour, the number of calls per month could be approximately estimated as 22 x 50,000/10% = 22 x 500,000 = 11,000,000 calls. This relationship must be stated in the performance requirements if any requirement relies on it.

Lilja has identified the following useful properties of performance metrics [7]. Among these are linearity (meaning that the performance of a system improves by the same ratio as the metric describing it), reliability (meaning that System A outperforms System B when the metric indicates that it does), repeatability (meaning that if the same experiment is run more than once under identical load conditions with identical configurations, the resulting metric will always have the same value), ease of measurement, consistency (meaning that the metric has the same meaning across all systems), and independence (meaning that it does not reflect the bias of any stakeholder).

More than one metric may be needed to describe the performance of a system. For on-line transaction processing systems, such as a brokerage system or an online airline reservation system, the metrics of interest will be the response times and transaction rates for each type of transaction, usually measured in the hour when the traffic is heaviest. The transaction loss rate, i.e., the fraction of submitted transactions that were not completed for whatever reason, is another measure of performance. It should be very low. We see immediately that one performance metric on its own is not sufficient to tell us about the performance of a system or about the quality of service it provides. The tendency to fixate on a single number or to focus too much on a single metric, termed *mononumerosis* by Odlyzsko [11], can result in a poor design or purchasing decision, because the chosen metric may not reflect critical system characteristics that are described by other metrics. One cannot rely on a single number to tell us the whole story about the performance of a system. For example, a low system response time may be accompanied by a high transaction loss rate, because the loss rate reduces the waiting times of the transactions that have not been lost, or because lost transactions may have been more likely to suffer long delays.

4. GUIDELINES FOR SPECIFYING PERFORMANCE REQUIREMENTS

The criteria for performance requirements are a superset of those in [6] for functional requirements. In particular, like functional requirements, performance requirements must be unambiguous, traceable, verifiable, complete, and correct. Additional criteria relate to the quantitative nature of performance requirements. To be useful, they must be written in measurable terms, expressed in correct statistical terms, and written in terms of one or more metrics that are informative about the problem domain. They must also be written in terms of metrics suitable for the time scale within which the system must respond to stimuli. In addition, the requirements must be mathematically consistent. Finally, all performance requirements must be linked to business, regulatory, and engineering needs. We now elaborate on each of these criteria in turn.

4.1 Unambiguousness

First and foremost, a performance requirement must be unambiguous. Ambiguity arises primarily from a poor choice of wording, but it can also arise from a poor choice of metrics.

- **Example 1.** "The response times shall be less than 5 seconds 95% of the time."

This requirement is ambiguous. It opens the question of whether this must be true during 95% of the busy hour, 95% of the busiest 5 minutes of the busy hour (both of which may be hard to satisfy), or during 95% of the year (which might be easy to satisfy if quiet periods are included in the average). In any case, the response time is a sampled discrete observation, not a quantity averaged over time. Consider an alternative formulation:

- "The average response time shall be 2 seconds or less in each 5-minute period beginning on the hour. Ninety-five percent of all response times shall be less than 5 seconds."

This requirement is very specific as to the periods in which averages will be collected, as well as to the probability of a sampled response time exceeding a specific value.

- **Example 2.** "The system shall support all submitted transactions."

Requiring that a system shall support all submitted transactions is ambiguous, because

1. there is no statement of the rate at which transactions occur,
2. there no statement of what the transactions do,
3. there is no explicit definition of the term "support".

Instead, one might state that the submitted rate of type A transactions is 5 per second, or (equivalently) 60 per minute. If this requirement is coupled with an unambiguous response time requirement such as that given in Example 1, and a further requirement that no errors occur while the transactions are being handled, we may be able to say that the transaction rate is being *supported* if the response time and transaction loss rate requirements are also met. We may also be able to say that a desired transaction rate is *sustainable* all resources in the system are at utilization levels below a stated average utilization that is less than saturation (e.g., 70%) to allow room for spikes in activity when this rate occurs.

4.2 Measurability

A well specified performance requirement must be expressed in terms of quantities that are measurable. If the source of the measurement is not known or is not trustworthy, the requirement will be unenforceable. Therefore, it must be possible to obtain the values of the metric(s) in which the requirement is expressed. To ensure this, the source of the data involved in the requirement should be specified alongside the requirement itself. The source of the data could be a measurement tool embedded in the operating system, a load generator, or a counter generated by the application or one of its supporting platforms, such as an application server or database management system. A performance requirement should not be adopted if it cannot be verified and enforced by measurement.

- **Example 3**. The average, minimum, and maximum response times during an observation interval may be obtained from a commercial load generator, together with a count of the number of attempted, successful, and failed transactions of each type, but only if the load generator is set up to collect them.
- **Example 4**. The sample variance of the response times can only be obtained if the load generator also collects the sum of the squared response times during each observation interval, or if all response times have been logged, provided always that at least two response times have been collected.

4.3 Verifiability

According to [6], a requirement is verifiable "…if, and only if, there exists some finite cost-effective process with which a person or machine can check that the software product meets the requirement. In general any ambiguous requirement is not verifiable." For performance requirements, this means that the performance requirement is testable, consistent, unambiguous, measurable, and consistent with all other performance and functional requirements pertaining to the system of interest. Where a performance requirement is inherently untestable, such as freedom from deadlock, a procedure should be specified for determining that the design fails to meet at least one of the three necessary conditions for deadlock. These are circular waiting for a resource, mutual exclusion from a resource, and nonpreemption of a resource [3]. On the other hand, if deadlock happens to occur during performance testing, we know that the requirement for freedom from it cannot be met.

4.4 Completeness

A performance requirement is complete if its parameters are fully specified, if it is unambiguous, and if its context is fully specified. A requirement that specifies that a system shall be able to process 50,000 transactions per month is incomplete because the type of transaction has not been specified, the parameters of the transaction have not been specified, and the context has not been specified. In particular, to be able to test the requirement, we have to know how many transactions are requested in the peak hour, and then have some context for inferring that the peak hourly transaction rate is functionally related to the number of transactions per month. We also have to define a performance requirement for the acceptable time to complete the transaction.

4.5 Correctness

In addition to being correct within the context of the application to which it refers, a performance requirement is correct only if it is specified in measurable terms, is unambiguous, and is mathematically consistent with other requirements. In addition, it must be specified with respect to the time scale for which engineering steps must be taken.

4.6 Mathematical Consistency

There are multiple aspects to the mathematical consistency of performance requirements.

- Performance requirements must be mathematically consistent with one another. To verify consistency, one must ensure that no inference can be drawn from any requirement that would conflict with any other requirement. Inferences could be drawn through the use of models. They could also be drawn by deriving an implied requirement from a stated one. If the implied requirement is inconsistent with other requirements, so is the source requirement.
- Each performance requirement must be consistent with stated performance assumptions, e.g., about the traffic conditions and engineering constraints. For example, a message round trip time should be less than the timeout interval, while the produce of the processing time and the system throughput must be less than 100% so that the CPU is not saturated.
- The performance requirements must not specify combinations of loads and anticipated service times that make it unachievable. This will happen if the product of the offered traffic rate and the anticipated average service time

of any device is greater than the number of devices acting in parallel (usually one).

4.7 Testability

We desire that all performance requirements be testable. Testability is closely related to measurability. If a metric mentioned in a performance requirement cannot be measured, the requirement cannot be tested, either.

Not all performance requirements are directly linked to the ability to attain specific values for metrics, though. Moreover, such requirements may be very difficult to test. For example, as discussed above, freedom from deadlock must be verified from the system structure. Since the potential for deadlock can be masked under light loads, and since testing for freedom from deadlock involves the enumeration of all execution paths, freedom from deadlock is not verifiable by performance testing alone.

4.8 Traceability

Like functional requirements, performance requirements must be traceable. Traceability answers questions like the following:

1. Why has this performance requirement been specified?
2. To what business need does the performance requirement respond?
3. To what engineering needs does the performance requirement respond?
4. Does the performance requirement help us conform to a government or industrial regulation?
5. Is the requirement consistent with industrial norms? Is it derived from industrial norms?
6. Who proposed the requirement?
7. If this requirement is based on a mathematical derivation or model, the parameters should be listed and a reference to the model provided.
8. If this requirement is based on the outputs of a load model, a reference and pointer to the load model should be provided, together with the corresponding version number and date of issue.

4.9 Linkage to Business and Engineering Needs

All performance requirements must be linked to business and engineering needs. Linking to a business need reduces the risk of engineering the system to meet a requirement that is unnecessarily stringent, while linking to an engineering need helps us to understand why the requirement was specified in the first place. An example of a business need is the desire to provide a competitive differentiator from a slower product. An example of an engineering need is that a TCP packet must be acknowledged within a certain time interval to prevent timeouts. Another example of an engineering need is the standards requirement that an alarm be delivered to a console and/or sounded within a maximum amount of time from that at which the corresponding problem was detected [9].

5. QUALITATIVE ATTRIBUTES RELATED TO PERFORMANCE

Performance requirements may contain a statement of the form "The system shall be scalable." All too often, there is no mention of the dimension with respect to which the system should be scaled, or the extent to which the system might be scaled in the future. Absent these criteria for scalability, testers will not know how to verify that the system is indeed scalable, and product managers and sales engineers will not be able to manage customer expectations about the ability of the system to be expanded. Characteristics for scalability, such as load scalability, space-time scalability, space scalability, and structural scalability are described in [2]. Examples of the corresponding dimensions include transaction rates, the ability to exploit parallelism, storage available to users and the operating environment, and constraints imposed by the size of the address space.

Stability is a quality attribute that is also related to scalability. If the system runs smoothly when N objects are present but crashes when N+1 objects are present, the scalability of the system is limited by the number of objects the system can support. Clearly, the number of objects the system can support is a dimension of scalability that is limited in this case.

Stability or a tendency to instability are also indicated by characteristics of the performance metrics. For example, during a prolonged period when the average offered transaction rate is constant, one expects (a) that the completion rate to be equal to the transaction rate, (b) that average resource utilizations will be approximately constant, (c) that average response times will be approximately constant, and (d) that memory occupancy will be approximately constant. Performance requirements for these characteristics should be specified. Failure to meet them in performance tests or production should be cause for an investigation. Upward trends in any or all these measures is an indication of saturation or an oncoming crash. In particular, if memory occupancy is increasing, there may be a memory leak that could lead to a system crash.

6. DERIVED AND IMPLICIT PERFORMANCE REQUIREMENTS

While performance requirements about transaction rates, throughputs, and response times are often explicitly stated, consequent requirements on subsystems, including object pool size and memory usage are not. If they are not explicitly stated, they must be derived from the quantities that are given to ensure system stability and to ensure that sufficient numbers of concurrent activities can be supported. In the author's experience, an astute developer and/or tester may ask the performance engineer to specify the maximum size of the object pool so that testing can be done accordingly.

As an example, suppose that a transaction will be dropped if an object pool is exhausted. The required transaction response time may be thought of as an average value for the holding time, while the transaction rate multiplied by the number of times an object will be acquired and released by the transaction. If we require that the probability of object pool exhaustion is 10^{-6} or less, we can approximately size the object pool to achieve this requirement using the Erlang loss formula [4]. The calculated object pool size is the derived requirement needed to achieve the desired probability of pool exhaustion. While the loss probability requirement is inherently hard to test because losses should not occur, the ability to store the desired number of objects is easily tested in principle, provided that the test harness is capable of doing so.

In this context, it is worth noting that freedom from deadlock is always an implicit requirement. It can be derived from any requirement that specifies or implies a non-zero throughput, because a system in deadlock has zero throughput. Freedom from deadlock is a prerequisite for system stability.

7. PATTERNS AND ANTI-PATTERNS IN PERFORMANCE REQUIREMENTS

While performance requirements for specialized embedded systems may take unusual, domain-specific forms, those for transaction-oriented systems tend to fall into patterns for average response time, throughput, and number of supported users. We have already seen some examples of these in the foregoing. Smith and Williams have used the term *performance anti-pattern* to describe an aspect of system structure or algorithmic behaviour that leads to poor performance [13]. We shall use the term *performance requirements anti-pattern* to denote a form of performance requirement that is ambiguous at best and misleading at worst. Anti-patterns are to be avoided, even if they express sentiments that are laudable. We illustrate patterns and anti-patterns with examples encountered by the author.

7.1 Response Time Pattern and Anti-Pattern

The following is an ill-formulated performance requirement.

1. *Ideally, the response time shall be at most one second.*
2. *The response time shall be at most 2 seconds.*

This requirement is problematic. The term "ideally" expresses a sentiment, but does not describe something that is attainable. The two parts of the requirement are mutually inconsistent. The occurrence of a single response time in excess of two seconds would mean that the requirement had not been met. Nothing is stated about when or how often the response time requirement must be met. If a sentiment like that in the first part of the requirement must be documented, it is best to place it in a section on supporting commentary rather than in the body of the requirement itself.

We propose a formulation that expresses the same sentiment while being measurable and testable.

1. *The average response time during the busy hour shall be 1 second.*
2. *99% of all response times shall be less than 2 seconds during the busy hour.*
3. *Both requirements shall be met simultaneously.*

The wording in parts 1 and 2 of this requirement reflects the fact that the average response time is a sample statistic rather than a time-averaged statistic. Notice also that the second part of the requirement does not say that the response time shall be less than 2 seconds 99% of the time, since that would suggest averaging over time rather than over the observed values of the response time.

7.2 "…all the time/…of the time" Anti-Pattern

Were a requirement to say that the response time should be less than 2 seconds 99% of the time, we would have to clarify the requirement by asking whether the requirement for the average response time would be met for 0.99x3600=3564 seconds in every hour, or during some fraction of the year, or some other time interval. The problem may be illustrated by a quote from former President George W. Bush: "I talk to General Petraeus all the time. I say 'all the time' -- weekly; that's all the time – …"[8]. The quantification is ambiguous because the time scale and frequency of interaction are unspecified, and because one cannot tell from colloquial use whether the "…all the time" or "…of the time" refers to a sample statistic such as average response time, a time-averaged statistic such as utilization or queue length, or to a frequency, such as the number of events per second or even the number of communications between a president and a general per month.

7.3 Resource Utilization Anti-Pattern

A requirement that states that the CPU utilization shall be 60% is erroneous because the resource utilization depends on the hardware and on the volume of activity. The desired response time and throughput requirements might well be met at higher utilizations. Furthermore, the requirement would fail to be met under light loads, which is absurd. When confronted with a requirement like this, the performance engineer could ask whether the stakeholder who originated the requirement is concerned about overload, and then offer to reformulate the requirement as an upper bound on processor utilization. Doing so helps to ensure that the system will be able to gracefully deal with transients that could cause the utilization to briefly exceed the stated level under normal conditions. It is entirely appropriate to state a resource utilization requirement of the form "The average utilization of resource X shall be less than Y% in the peak hour." For single server resources, Y might be set to 70%. For a pair of parallel servers in which one acts as a backup for the other, it is appropriate to state that the utilizations of individual processors must not exceed 40%, so that the maximum load on one of them after a failover would be no more than 80%. Anything higher than that could result in system saturation.

8. PERFORMANCE REQUIREMENTS GATHERING

As with functional requirements, the gathering of performance requirements entails interviewing stakeholders from many different teams within the customer and supplier organizations. In the author's experience, the set of stakeholders can include product managers and sales engineers, because they identify the market segments for a system, including customers for large-scale and small-scale systems. Any pertinent regulations that could affect performance requirements must also be identified, such as fire codes in the case of alarm systems. The set of stakeholders also includes architects, designers, developers, and testers. It is important to interview the architects and developers, because they may propose the use of technologies that are incapable of meeting the envisaged system demand.

Stakeholders may be reluctant to commit to a particular set of estimates of demand for system usage because Customer A may argue that his organisation's load is not like Customer B's. For example, the work mix of a small rural clinic may be very different from that of a large hospital using similar sorts of computer-controlled diagnostic equipment for different purposes. Even the workloads of hospitals with similar numbers of beds may differ, because one hospital might specialize in orthopaedics while the other only does cancer care. Their fire alarm systems may be quite different, too, because of the nature of what is stored. Despite these disparities, performance requirements specification and testing should not be avoided. Instead, the project team should resort to the use of a set of *reference scenarios* reflecting standardized mixes of activities. The reference scenarios might be agreed to by product managers and/or sales engineers, and then mapped to the corresponding activities in the computer system, with corresponding workloads. The frequency and delay requirements of these activities form the body of the performance requirements. Under no circumstances should performance requirements be reduced to a single number,

because doing so will mask potential complexities and obscure any possibilities for tradeoffs.

9. PERFORMANCE REQUIREMENTS PITFALL: TRANSITION FROM A LEGACY SYSTEM TO A NEW SYSTEM

When transitioning from a legacy system to a new one, it is easy to overlook subtle changes in functionality that might affect the way performance requirements should be formulated. The author encountered this pitfall when transitioning from a 1940s vintage 35mm rangefinder camera to a modern point and shoot digital camera. With the old camera, pressing the shutter button causes the subject to be captured pretty much in the state seen by the user. In this case, the object was a walking cow with a bell hanging from its collar. The digital camera took so much time to capture the image that the resulting photo included the cow's udder, but not the bell. The difficulty was that the shutter reaction time with the digital camera included autofocus and exposure setting. With the vintage camera, these would have been done manually in advance of the shutter being released. The problem occurred because the photographer simply assumed that the digital camera would have the same shutter reaction time as the vintage camera. It does not, and the unexpected image was the result. One might ask whether the comparison of the shutter reaction times is fair, given that the digital camera does so much more when the button is pressed. The answer is that a comparison should reflect expectations of the functionality that will be implemented, and that the user should plan the shot accordingly.

10. STRUCTURE OF A PERFORMANCE REQUIREMENTS DOCUMENT

The structure of a performance requirements document we recommend is quite similar to that recommended in [6] for functional requirements. A section on traffic assumptions specific to the domain should be included to reduce the risk of ambiguity or misunderstanding. This is especially important in a labour force with turnover. Reference work items and reference workloads are needed to establish the context for domain-specific metrics. A reference work item may be a particular kind of transaction or set of transactions and activities. A reference workload specifies the mix and volumes of the transactions and activities. A reference scenario might be a set of workloads, or a set of actions to be carried out upon the occurrence of a specific type of event. For instance, a reference scenario for a fire alarm system might be the occurrence of a fire that triggers summoning the fire brigade, the sounding of alarms, and the automated closure of a defined set of ventilators and doors. The performance metrics used in the requirements, especially those that are specific to the domain, should be defined and mapped to related system actions. The instrumentation used to gather the metrics should also be specified to the extent known, so that one can establish that a mechanism for verifying and enforcing the requirements exists. Figure 1 shows a possible outline for a performance requirements document.

1. Scope and Purpose
2. Intended Audience
3. References (including functional requirements spec)
4. Statement of Assumptions:
    a. Traffic assumptions Specific to the Domain
    b. Definition of reference work items and reference workloads and scenarios
    c. Criteria for load sustainability
    d. Definitions of metrics used in the requirements
    e. Instrumentation to gather the metrics for verification
5. Performance Requirements

Figure 1. Outline of a performance requirements document.

Table 1. Suggested fields of a performance requirements record.

1.	Requirement Number
2.	Title
3.	Statement of requirement
4.	Supporting commentary
5.	List of precedents, sources, standards
6.	Derivation of quantities
7.	List of dependent requirements
8.	List of assumptions and precedent performance requirements
9.	Sources of measurement data.
10.	Name of a subject matter expert on this requirement
11.	Indicator if the requirement is independently modifiable, or if not, why not.
12.	Indicator that the requirement is traceable.
13.	Indicator that the requirement is unambiguous, or if not, why not.
14.	Indicator that the requirement is correct, or, if not, why not.
15.	Indicator that the requirement is complete, and if not, why not.
16.	Indicator that the requirement has passed or failed review, and why.

11. STRUCTURE OF A PERFORMANCE REQUIREMENT

The fields of a performance requirement record suggested below reflect many of the concerns we have described above. Some, like the list of precedents, sources, and standards, are intended to provide traceability. Separating supporting commentary from the statement of the requirement reduces the risk of ambiguity while providing an opportunity to document some of the reasoning behind the requirement and the requirement's purpose. Listing dependent and precedent performance requirements help one to see how requirements are intertwined. A possible list of records is shown in Table 1.

12. THE COMMERCIAL SENSITIVITY OF PERFORMANCE REQUIREMENTS

Disclaimer: This section does not contain legal advice. You should seek the advice of legal counsel when drafting any agreements or documents incorporated into agreements by reference. Legal obligations and practice may differ from one jurisdiction to another. The author is not a lawyer.

12.1 Confidentiality

A great deal can be inferred about the competitiveness of a product or the commercial position of the intended customer by examining performance requirements. For example, the ability of a network management system to handle traps at a given peak rate, combined with knowledge of the number of nodes to be managed and the peak polling rate can tell us about the intended market segment of the product while nourishing speculation about the product's feature set, or even about the nature of the site the system is intended to support. This can affect price negotiations between supplier and buyer, and perhaps the supplier's share price. Therefore, performance requirements and any contractual negotiations related to them should be treated as confidential and perhaps even covered by non-disclosure agreements (NDAs). The release of performance requirements and performance data outside a circle of individuals with a need to know should be handled with great care. Engineering, marketing, legal, and intellectual property departments should all be involved in setting up a formal process to release performance data to the public or to third parties under non-disclosure agreements.

12.2 System Performance and the Relationship Between Buyer and Supplier

Situations may arise in which the supplier has greater expertise in system performance than the buyer, or vice versa. In the author's experience, both are possible whether the buyer is a startup and the supplier is established, both are startups, both are established, or the supplier is a startup and the buyer is established. In any of these cases, transparency and adherence to commonly accepted guidelines for writing requirements, such as those prescribed by IEEE Std 830-1998 for software requirements documents [6] will go a long way to preventing misunderstandings and disputes regarding performance requirements and the interpretation of performance test results.

13. MANAGING PERFORMANCE REQUIREMENTS

Performance requirements play a role in every stage of the software lifecycle, whether the lifecycle is managed using a waterfall process, an Agile process, or something else. To facilitate access by the stakeholders, performance requirements should be centrally stored, perhaps in the same system that is used to store functional requirements. To ensure that performance considerations do not fall into a crack, it is essential that an individual be nominated as their owner, and that this individual be visibly mandated and empowered to communicate with all project stakeholders about performance issues.

In addition to addressing performance concerns, the performance requirements owner will be responsible for managing change control, requirements traceability, as well as ensuring that every change or addition is linked to business and engineering needs. The owner will also play a pivotal role in mediating between different groups of stakeholders when performance requirements are negotiated and written, including architects, designers, and perhaps even lawyers. Involvement with the latter is necessary to ensure that contracted levels of performance are described in measurable terms. If performance requirements are changed, the performance requirements owner must ensure that the changes are understood by architects, developers, and sales engineers, so that the necessary changes to architecture, implementation, and appropriate commitments to customers can be made.

14. CONCLUSION

The foregoing discussion has covered a wide range of topics related to performance requirements. Careful wording of performance requirements is necessary to ensure verifiability and testability. Ambiguity and confusion occur when a performance requirement contains inconsistencies, or when it is inconsistent with other requirements or with standards documents. Ambiguous and otherwise ill-specified requirements lead to time wasted trying to sort out what they mean. The absence of performance requirements can lead to disagreements among stakeholders about performance expectations. We have proposed guidelines for writing and managing performance requirements that are consistent with those for functional requirements. The performance requirements must be formulated in terms of metrics whose values can be measured in testing and in production. The metrics must be relevant to the application domain. We have also shown how reference scenarios and reference workloads can be used to steer stakeholders towards a clear baseline when the possible set of performance requirements is very large. Our experience suggests that adherence to the practices here can be used to avoid many performance pitfalls, while aiding in the smooth application of performance engineering principles in the software lifecycle.

15. ACKNOWLEDGMENTS

The author has benefited from useful discussions with Alberto Avritzer, Brian Berenbach, Dan Paulish, and Bob Schwanke of Siemens Corporate Research, as well as from the experience of teaching performance requirements practices in internal training courses.

16. REFERENCES

[1] Bass, Len, Robert L. Nord, William Wood, David Zubrow: Risk Themes Discovered through Architecture Evaluations. WICSA 2007, Mumbai, January 2007.

[2] Bondi, A. B.: Characteristics of Scalability and Their Impact on Performance. Proc. WOSP2000 (Workshop on Software Performance), 195-203, Ottawa, Canada, September 2000.

[3] Coffman, E., and P. J. Denning. Operating Systems Theory. Prentice Hall, 1973.

[4] Cooper, R. B., Introduction to Queueing Theory, Second Edition, North Holland, 1981.

[5] Ho, C.-W.; Johnson, M.J.; Williams, L., and Maximilen, E.M.: On agile performance requirements specification and testing. Proc. Agile Conference, Minneapolis, 2006,

[6] IEEE Std 830-1998 IEEE Recommended Practice for Software Requirements Specifications –Description

[7] Lilja, D. J. *Measuring Computer Performance: a Practitioner's Guide. Cambridge University Press,* 2000.

[8] http://www.prnewswire.com/cgi-bin/stories.pl?ACCT=104&STORY=/www/story/06-28-2007/0004617850&EDATE=

[9] NFPA 72, National Fire Alarm Code ®, 2007 Edition, NFPA, Quincy, MA 02169-7471.

[10] Nixon, B.A.: Managing performance requirements for information systems. Proc. First WOSP 1998, 131-144, 1998.

[11] Odlyszko, A.M., CMG Magazine, 2000.

[12] R. F. Rey (ed)., Engineering and Operations in the Bell System, AT&T Bell Laboratories, 1983.

[13] Smith, Connie U., and Lloyd G. Williams. Performance Solutions: a Practical Guide to Creating Responsive, Scalable Software. Addison-Wesley, 2000.

[14] Webster's Ninth New Collegiate Dictionary, Merriam Webster, 1988.