Performance Testing an Enterprise System from the Customer's Point of View

Ulrike Dowie, Arne Gehler, Prof. Dr. Georg Herzwurm

University of Stuttgart, Siemens AG
Breitscheidstr. 2c, 70174 Stuttgart, Germany
Phone: +49-711-121-2385, Fax: +49-711-121-2388

dowie@wi.uni-stuttgart.de, arne.gehler@siemens.com; herzwurm@wi.uni-stuttgart.de
Performance Testing an Enterprise System from the Customer's Point of View

Abstract

Different perspectives between software producer and customer of a software application’s performance are portrayed as one of the reasons for software performance problems. However, software producer and customer can cooperate to overcome these differences. A case study is presented in this paper that describes the process which led to the successful implementation of a consolidation and reporting system which satisfied the customer’s functional and performance requirements on the software. The way of cooperation between the software producer and the customer is proposed to be applied in future software development projects by deducting generalized lessons learned from this case study.

Key words

Software performance, performance test, software requirements, performance evaluation.

1 Introduction

“Software performance is one of the main attributes of software quality” (D’Ambrogio and Iazeolla, 2005, p. 127).

A lot of research has been done on how to model, measure, analyze and predict system performance (e.g., Ould-Khaoua et al., 2005, Cortellessa et al., 2005, Avritzer and Weyuker, 2004). In scientific publications it has been recognized as being of vital importance throughout the entire software lifecycle (Ould-Khaoua et al., 2005). Software producers, as well, are conscious of the customers’ awareness regarding system performance (e.g., the 3rd edition of “SAP R/3 Performance Optimization” has been published by SAP, 2nd edition of “Oracle Performance Tuning” published in 1996). Especially regarding financial information, changing legal requirements and demands of the financial community turn rapid availability of current data into a decisive competitive factor (Werner et al., 2004). Yet still there are numerous cases of customer dissatisfaction due to system performance problems (Avritzer and Weyuker, 2004; Zadrozny et al., 2002).

In this paper we will describe what we assume to be a major reason for these problems: the differing perspectives on system performance between the software producer and the end users. Quality assurance activities, particularly performance testing are aimed at avoiding performance problems if employed early enough. However, as will be shown in a case study, considerable effort has to be spent by both software producer and customer in order to produce and be able to employ software that fulfills the customers’ performance requirements. If these efforts are aligned and put under way from the onset of the project, performance can be built into the software and the application can perform as expected by the customers as soon as it is implemented.

Apart from the fact that a satisfactory level of performance can be created, the case study also serves as a proposal on how to go about software development to
this end. In particular, performance testing proved to be very valuable for improving the software’s performance.

To facilitate applying the procedure described in the case study for future software development, lessons learned will be summarized. Finally, conclusions will be drawn on a more general level and future research will be proposed. Their intention is to give ideas what can be done, by software producers and customers alike, to make meeting customers’ performance requirements the rule, not an exception, before the system in question is being released.

2 Perspectives on software performance

Since performance is one of the characteristics that software quality is composed of as defined by the ISO in (ISO/IEC 9126-1:2001), perspectives on performance can be distinguished the same way as perspectives on software quality as a whole:

1. the software producer’s point of view, and
2. the customer’s point of view.

These perspectives form part of the software quality model presented in (ISO/IEC 9126-1:2001): internal quality attributes of the software product which represent the developers’ view, external quality attributes of the software product that become observable when executing the application in the testing phase in a simulated environment, and quality in use attributes that represent the users’ point of view on the software. The quality in use attributes manifest themselves to the user when employing the software application, and can be defined as the combined effect of the software quality characteristics (ISO/IEC 9126-1:2001, p. 1).

Perspective in this paper’s context includes several aspects:

1. What is perceived as performance?
2. How can it be tested?
3. How can it be improved?

The software performance the software producer intends to develop - if this is one of the stated development goals - is a characteristic of the software application by itself. The developers can be regarded as manufacturers of the software (Bach, 1995). As such, their intention is to design and implement an application such that its performance (as one part of the software’s quality) meets the specified performance aims. The application’s performance thus can be perceived by checking whether design and code match the specification. The customer in contrast is interested in carrying out the tasks the software, e.g. an enterprise system is supposed to support in an efficient way. He therefore looks at an entire system’s performance, including application, hardware, network, databases, servers, etc. Thus, what a customer means by performance of an enterprise system has to be defined individually (Zadrozny et al., 2002, p. 3), since it depends on a number of factors. Apart from the components used in the system, the application usage (see section 3) is of great influence. This includes for example the type and amount of input data, the number of simultaneously working users and the frequency and intensity of use of individual functions. Finally, the customizing done in the system to reflect and support the customer’s business processes has major influence on the performance perceived by the customer.
Performance can be tested both by software producer and customer. The testing aims, however, may differ, due to the development phase in which the tests are carried out, and due to the degree in which the test data and test environment represent realistic conditions. This will be explained in further detail in the next section.

Performance improvement can be done by software producer and customer, yet in different ways and to very different degrees. In line with his perspective on the software application, the software producer has the possibility to change design and code of the application. In practice, however, once the design is finished, design changes become more and more difficult the further implementation has progressed. Depending on the part of the software to be changed, especially the number of processes using the piece of code to be changed and the amount of code affected by such a change, changes to the code can be made even after delivery of the software. In this case, the corrected code gets distributed to the customers who already employ the software (e.g. in the form of a support package).

Customers, on the other hand, typically have reduced possibilities to change the application, particularly regarding standard software that they have licensed. They can, however, change the customizing that was done to the application, i.e. the adaptation of the application to their specific business processes, data etc. Another possibility for the customer to improve a software system's performance is to exchange components such as the database or to add more hardware resources. Corresponding to the customer’s perspective on software performance, all of these possibilities treat the system (consisting of various components) as the object to undergo performance improvement, not the application by itself.

3 Performance testing and other quality assurance activities to improve software performance

Testing and other quality assurance activities can serve to improve the quality in general and the performance of an application in particular (Khoshgoftaar et al., 2005, p. 111). Tests do this by allowing to measure performance and decide whether performance goals or requirements have been reached resp. fulfilled. Thus, testing provides information on the level of quality of a software product and the risks involved when using the software (Koomen et al., 2000, p. 13).

Running performance tests allows measuring how long the application needs to process a user's request that represents a certain use case (Spillner and Linz, 2004, p. 60). A specific type of performance tests called load tests serve to determine the application’s performance under varying workloads, i.e. amount of data to be processed and number of concurrent users, that is, users working in the system simultaneously.

The key goal of performance testing is to uncover performance problems that manifest themselves when the system is run under specific workloads (Avritzer and Weyuker, 2004, p. 1072). Unexpected behaviours of the system are revealed that make the software product miss the quality requirements. Thus, this quality assurance activity is capable of delivering information to evaluate the achieved level of quality in use, i.e. the application’s performance as perceived by the user.

Another goal of performance testing can be to exploit the potential of a system (Ould-Khaoua et al., 2005, p. 2). Whereas discovering performance problems in order to improve the software at hand is of interest to both software producer and customer, determining the potential of a system is much more relevant to the software producer since he can use this information to market the software.
Several steps have to be carried out before running the performance tests (Zadrozny et al., 2002, p. 12f):

1. Performance criteria have to be defined (i.e., relevant metrics have to be identified and realistic targets for them).

2. Application usage has to be accurately simulated, that is, test conditions have to be defined that mimic reality as closely as possible, in order to evaluate the software against the degree of performance the users perceive and require.

3. A sampling method and associated metrics have to be defined. This means that depending on the testing objective, e.g. testing existing applications for capacity planning or architectural configuration, or testing to explore programming best practices, the appropriate method has to be chosen.

Determination of a realistic application usage, also called operational profile, is the most important of these steps. An operational profile consists of a description of how the system is used when it is operational in the field (Avritzer and Weyuker, 2005, p. 1072). This important task is also the most difficult one when it comes to developing new software applications where there is no experience regarding typical application usage. In this case, application usage has to be determined indirectly through interviews with system engineers and other experts in the task area the application is supposed to support.

The sort of performance testing which requires a system set-up and defined application usage can be carried out both by the software producer and the customer. What is different in each case is the tested application usage. Whereas the customers by definition know how they will want to employ the software, the software producer can only test what he assumes to be a realistic usage of the application.

Performance tests can also be carried out in order to choose between different design alternatives. This sort of performance testing generally applies to the software producer only. To select the design alternative best suited for the application being developed, prototypes have to be developed on which the performance tests are run. This implies that the effort to build the prototypes has to be taken into consideration, and that much of the coding and design results achieved at this point might become obsolete. For these reasons, performance testing as a tool to determine the actual costs or benefits of a design alternative (Zadrozny et al., 2002) is only recommendable when there’s little experience with the technologies involved, and when building a prototype that contains enough functionality to deliver valuable information when performance tested, requires an acceptable amount of effort.

Improving an application’s performance in the testing phase is more expensive than in earlier phases like design or implementation (e.g., D'Ambrogio and Iazeolla, 2004, p. 127). Therefore, other techniques to analyze and evaluate performance come into play that are applicable as early as the design phase, such as architecture-based analysis (Gokhale et al., 2004) and performance modelling (Avritzer and Weyuker, 2004).

1. Architecture-based analysis offers information upon which to identify performance-critical components and the influence individual components have on the performance of the application as a whole.
Performance modelling as one specific variant of simulation modelling is used to represent known features concerning the application’s performance in an abstract way. Insights are gained from extrapolating from the modelled behaviour. However, extrapolation does not yield valid results for all sorts of performance tests when applications consisting of several sub-systems such as an enterprise system are concerned. For example, the sub-systems composing an enterprise system such as application server, database and operating system do not interact the same way across different user loads (Zadrozny et al., 2002, p. 19). Therefore, simulation as an activity cannot replace performance testing. Instead, performance testing and performance modelling are complementary in terms of the type of problems they tend to reveal. They should be used in conjunction with each other.

4 Case study: Performance testing a consolidation and reporting enterprise system

The following case study of a large implementation project (i.e., more than two years duration and more than 25 full-time employees) illustrates the benefits of early and continuous performance testing for both the software producer as well as the client. It clearly demonstrates

(1) the objectives of both parties in doing the performance testing,
(2) the steps that were undergone in order to fulfil these objectives as well as
(3) some of the issues that were encountered in the testing process.

In this particular case, the system to be implemented was a financial consolidation and reporting system. The client, a large multi-national high-tech company, was ‘pilot’ user of this new release of a standard software product being developed. ‘Pilot’ here refers to the client being able to test the software under development and to influence development according to their specific requirements. Standard software refers to packaged software applications that are sold to different customers with no or with relatively little modifications compared to the application’s size (Hansen and Neumann, 2001, p. 152). Among the goals of this implementation project were

(1) to fully replace the existing consolidation and reporting system,
(2) to offer additional consolidation and reporting functionality (e.g., two-dimensional or matrix consolidation)
(3) to realign external and internal reporting data and
(4) to improve reporting flexibility and performance by setting up a new data warehouse management system.

When the implementation project was initialized, an agreement was made between the software producer and the client to cooperate in the development of the enterprise system and to follow an iterative development cycle. The producer agreed to roll out prototype versions of the system to the client, while the client agreed to thoroughly test the prototypes and to deliver input and feedback about the necessary performance and missing functionality early in the project and at certain milestones.

An important activity the client was doing throughout the project phases was to set up different kinds of performance tests. These tests were conducted in order to reach the following goals:
To set up a hardware environment capable of running the new system at a comparable speed to the system which was to be replaced and to discover performance bottlenecks early on in the development phase in order to easily eliminate or minimize those bottlenecks.

To be able to measure and compare the performance of the old and the new system, various metrics were analyzed. The selected metrics should be an indicator of performance as perceived by the end users (as opposed to the administrators’ or developers’ perspective). They had to be measurable in the old as well as in the new system. System performance would be most important and noticeable when some functionality was carried out that was used frequently by a considerable number of users. Thus, the user profile combined with the number of users calling the same function simultaneously had to be determined. Instead of determining an average value, each month’s peak times of system usage were analyzed. The greatest load in the old system occurred when the consolidation entities would enter their monthly reporting data. Since the functions they would call (data upload, various aggregating algorithms, etc.) were no standard features of the system but customer-specific and of different complexity, the metric should measure number of functions completed in one hour multiplied by number of users calling them. This would guarantee that complex functions as well as simpler ones would enter the measurement alike.

Analysis of data entry into the old system also showed that in a one-hour time span during the peak times, a large number of diverse consolidation entities would carry out data entry functions. This metric had the advantage that the new system’s expected behaviour could be precisely stated under the known user-load: The number of functions completed in the selected time slot had to be no less than in the old system under the same user load. For measuring and interpreting this measure, there was no need to know how the functionalities would be implemented in the new system.

The research that went into defining this metric was the basis of the hardware sizing for the new system. The three major issues of this part of the project were

1. the fact that the application as well as its complex customizing were not nearly finalized and
2. the fact that because of the long duration of the project, only very little hardware would be bought in the early phases (and used but obsolete hardware could not be returned at the end of the project) and
3. the fact that no usable test-data was available to run complex performance tests.

In order to deal with these shortcomings, a couple of assumptions were agreed upon and an iterative process was used in performance testing as well.

Based on the assumptions about the complexity of the processes in the new system, very complex test-data was created and fine-tuned to represent an entire fiscal year. The basis for the test-data was derived from the consolidation logic of the new system and the interdependencies and the complexity of the real data in the old system.

An iterative approach was used to redo the performance testing with every new prototype that was delivered by the software producer. For each iteration,

1. additional hardware
2. better test-data and
3. closer-to-final customizing was being used.
With each cycle through the performance testing, the assumptions were hardened and the knowledge about the final performance improved.

As part of the performance testing, the software producer and the client were able to discover performance bottlenecks that were only visible in certain high-load situations. For every phase, the software producer received a copy of the test-data as well as the customizing and was thus able to rerun the complete test-cycle several times to simulate those bottlenecks and propose solutions.

By the end of the project, both the software producer and the client gained major insight into the behaviour of this new release of the enterprise system. The client was able to go live with the new system in a timely fashion and the software producer was able to use the experience to improve the product. By working closely together, software producer and client both benefited from their efforts spent on requirements definition and testing.

5 Lessons learned from this case study

The case study presented is a rather untypical example of the process applied for developing standard software such as enterprise resource planning systems. However, its success as judged by software producer and client alike has proved the chosen cooperative and iterative development process to be worthwhile. From the experience gained we have deducted the following recommendations for development of enterprise systems that meet the customers’ performance requirements:

1. In the case study, the customer could indicate detailed, measurable performance requirements by comparing the new software to the former solution. This facilitated focusing the software development process on achieving the performance goals formulated as customer requirements. User requirements regarding performance (and other functional and non-functional quality attributes) of the application have to be taken into consideration from the start of development and they should form part of the specification. These requirements need to be as exactly defined as possible and should be measurable.

2. Delivery of several preliminary releases of the consolidation and reporting software and subsequent iterative performance tests by the customer in his system environment served very well to detect performance problems and solve them before employing the application in a productive environment. An iterative approach and frequent feedback loops between customer and software producer that include transferring performance test results help to design for and reach a performance level that satisfies the customers requirements.

3. Creating realistic test data and using productive data volumes for the performance tests greatly improved accuracy and validity of the test results in the case study presented. The hardware ordered based on the performance test results and the system performance delivered in the released product version left the customer very satisfied. Although considerable effort has to be spent to create amounts of realistic test data that mimic productive data loads, this effort is necessary in order to get useful and reliable test results. These are of great value to determine the
required hardware and eliminate performance bottlenecks from the enterprise system.

(4) Both software producer and customer felt it worthwhile to spend effort on determining (functional and) performance requirements as well as planning, running and analyzing the results of the performance tests. The software producer developed a product of improved performance and functionality that was focused on potential customers’ needs, thus creating a competitive advantage. The customer received a product that entirely fulfilled his requirements on functionality and performance.

Tight cooperation on a frequent basis between software producer and customer requires considerable effort but is highly beneficial to both parties.

(5) Although far from depicting a realistic working environment, the first performance tests carried out in a test environment with simplified, reduced customizing were an important first step for defining the necessary hardware to be ordered and finding performance shortcomings in the software.

Performance tests should be carried out as early as possible, even if application development on behalf of the software producer and customizing on behalf of the customer has not been finished.

6 Conclusions and proposal for future research

Despite different perspectives on software performance, there are possibilities for software producers and customers to align their efforts so that software applications are developed that satisfy their customers’ requirements on software performance. Both parties gain from this because customers receive software that fulfils their requirements and thus effectively supports their business tasks, and software producers avoid customer dissatisfaction which reduces support costs, and can offer a product that is characterized by “fitness for use” (Bach, 1995).

The case study presented differs from typical standard software development. Like in individual software development, the customer could define requirements, weigh and subsequently prioritize them before they became part of the requirements specification. In individual software development, the requirements would also become part of the contract with the software producer.

Although the benefits of this approach are well illustrated by the case study, this proceeding is generally not used in standard software development of which enterprise systems are a prominent example. Apparently, the customer is not known. However, when development of a standard software product is planned, there is presumably at least some idea about which sort of customer will use the software, and for what sort of purposes. In these cases, a representative group of (potential) customers could be interviewed regarding their individual preferences for functional and non-functional requirements of the software to be developed. The software producer then has to decide how to weigh the various customer representatives’ prioritized requirements, and create a requirements specification accordingly. This process, however, as well as the selection of the representative customers is a difficult task requiring experience. With the benefits of the proposed proceeding outlined by the case study, this paper can serve to motivate software producers to try this approach and find out whether they experience similar success.
The case study and the subsequent suggestions are one example of how to put the recommendations into practice that Capers Jones included in his “Summary of opportunities for software development”: Prototyping, Joint Applications Development and improvement of the partnership with the customer (Jones, 1997).

Scientific research could facilitate adoption of this approach. Research questions to be investigated include the following:

1. How can the customers’ expectations on quality characteristics such as performance be measured?
2. Is performance up to a certain level an implicitly expected software characteristic that the customer notices only when it is missing, and above that level a quality feature or excitement feature that the customer demands explicitly or is positively surprised when finding it in the software?
References


