

An Introduction to Performance Testing

Chih-wi Ho, Laurie Williams
North Carolina State University
dright@acm.org, williams@csc.ncsu.edu

Performance Testing is an important technique for ensuring a system will run as expected by the customer. In this chapter, we will explain the following:

- the role of performance requirements in performance testing
- information on how to do performance testing
- when to consider performance in the software development lifecycle

In a software system, *performance* is the degree to which a system or component accomplishes its designated functions within given constraints, such as speed, accuracy, or memory usage [12]. Performance is an external quality based upon user requirements [10] and the user's view of the operational system. Performance is also especially critical for real-time systems in which actions must complete within a specified time limit for correct operation [6] (i.e. the system will not operate correctly if the timing does not meet the specification, even if the user cannot perceive the performance problems). Performance may be described with the following indices:

- *Latency*: the time interval between the instant at which an instruction control unit issues a call for data and the instant at which the transfer of data has started [12]; the delay between request and completion of an operation [7].
- *Throughput*: the amount of work that can be performed by a computer system or component in a given period of time; for example the number of jobs in a day [12].
- *Resource consumption*: the amount of memory or disk space consumed by the application [16].

The performance of a system is highly visible to the software user. A system that runs too slow is likely to be rejected by all users. Therefore, software engineering teams must understand their customers' performance requirements and test their systems to ensure the system meets these requirements. Most of the products which fail in the market (after the release) are not system crashes or incorrect system responses, but performance degradation [19]. Data from 30 architectural reviews AT&T Labs indicated that 30% of the projects are at a medium or high risk of failure. In these architectural reviews, 93% of the performance-related project-affecting issues were found in these medium or high risk projects [20].

Performance testing is the testing conducted to evaluate the performance of a system with specified performance requirements [12]. During the process of performance testing, data must be collected. This data enables the accurate prediction of system behavior under varying workloads [1].

1. The Role of Requirements in Performance Testing

Customers' performance expectations are documented as performance requirements. A *performance requirement* is a requirement that imposes conditions on a functional requirement; for example, requirement which specify the speed, accuracy, or memory usage within which a given functionality must be performed [12]. As such, performance is considered a non-functional requirement¹. Non-functional requirements, such as performance, can easily be overlooked when customers and requirements elicitors focus too much on the functionality that the software development team needs to deliver. Poor performance costs the software industry millions of dollars annually in lost revenue, decreased productivity, increased development and hardware costs, and damaged customer relations [22]. As a result, software development teams need to focus on performance from requirements elicitation throughout development and into the testing phases.

Performance requirements can be specified qualitatively or quantitatively. Quantitative specifications are usually preferred because they are measurable and testable. Basili and Musa advocate that quantitative specification for the attributes of a final software product will lead to better software quality [3]. For performance requirements, Nixon suggests that both qualitative and quantitative specifications are needed, but different aspects are emphasized at different stages of development [16]. In the early stages, the development focus is on design decisions, and brief, qualitative specifications suffice for this purpose. At late stages, quantitative specifications are needed so that the performance of the final system can be evaluated with performance measurements.

Some example requirements, with an increasing amount of detail now follow:

- *The authentication process shall be completed quickly.*
- *After the user enters the user name and password, and clicks the Submit button on the Log In page, the response time for authentication and Main page rendering shall be within three seconds.*
- *After the user opens the Log In page, the user enters the valid user name and password, and clicks the Submit button. On average, this scenario happens 20 times per minute. After the user opens the Log In page, when the user enters the valid user name and password, and clicks the Submit button, the response time shall be less than 3 seconds 80% of the time.*
- *The system is running under the heaviest possible workloads [defined elsewhere]. The average response time for displaying the promotional message on the mobile tablet after a customer enters a lane where the promotional items are located shall be below 1 second.*

The first requirement is imprecise and not measurable/testable because no quantitative measures are provided. However, such a qualitative requirement may be appropriate early in the development lifecycle. The remaining three are testable – but the testing is more complex for each successive requirement. The difference between the third and

¹ Non functional requirements are not specifically concerned with the functionality of a system but place restrictions on the product being developed [13]

fourth descriptions is that, in the third one, the requirement describes how often a single service is requested by the users, while the fourth describes the request patterns of the whole system.

2. Performance Testing

Performance testing is usually via black box testing² done after software development and functionality testing is complete with the system tested for functional correctness. The true performance of a system cannot be assessed until all system elements are fully integrated [17], and the system is tested in a representative environment so that “real-world” loading and use can be evaluated. However, the performance of individual modules may be assessed as white-box tests³ to obtain an early warning of performance problems [11]. Additionally, performance test cases need to be written to specifically test the performance criteria rather than functional correctness [22]. Functional correctness can be ignored in performance testing, though. Some functional defects only surface under heavy workload. For example, if a program does not release the database connection, this problem does not show up if we just run the program once. At a heavy workload, the available database connections are quickly consumed. After no database connection is available, the program may generate an exception, and result in very short response time. If the functional correctness is not checked, the testing result might show good performance even though the system is not working properly.

Specified workloads, or a collection of requests, need to be generated for performance testing. Suppose we were developing a Web-based course registration system for a university with 5,000 students. We might expect 200 concurrent users for this system at the last days of course registration. Using 200 physical machines to simulate 200 concurrent users for performance testing is impractical. On the other hand, if all 200 requests are generated from a single machine, the machine will be busy context-switching among the threads or processes. As a result, the generated requests may not be the same as specified in the requirement. In a real performance testing environment, the workloads are generated from relatively limited resource. Five machines might be used to simulate 200 “virtual users” for this course registration system.

Figure 1 shows an architecture that is used in most performance testing tools. In this architecture, the software system is deployed on several servers, including maybe Web servers, database server, and so on. When performance testing starts, the centralized *controller* initializes the *load drivers* with the workload information. A load driver is a *software program that takes the workload information as input, and generates requests that mimic the user behavior* [18]. After the requests are generated and sent to the servers, the performance monitors installed on the servers observe the server behavior and record performance. A *monitor* is a *program that observes, supervises, or controls the activities of other programs* [21], such as its performance. The collected performance data may be sent back to the controller during testing, if the tool supports real-time performance

² Black box testing is testing that ignores the internal mechanism of a system or component and focuses solely on the outputs generated in response to selected inputs and execution conditions [12].

³ White box testing that takes into account the internal mechanism of a system or component [12].

Performance Testing

monitor. Otherwise, the data can be analyzed after performance testing is done. Using this architecture, workloads can be generated more precisely, and performance data collection has little effect on the performance of the system under test.

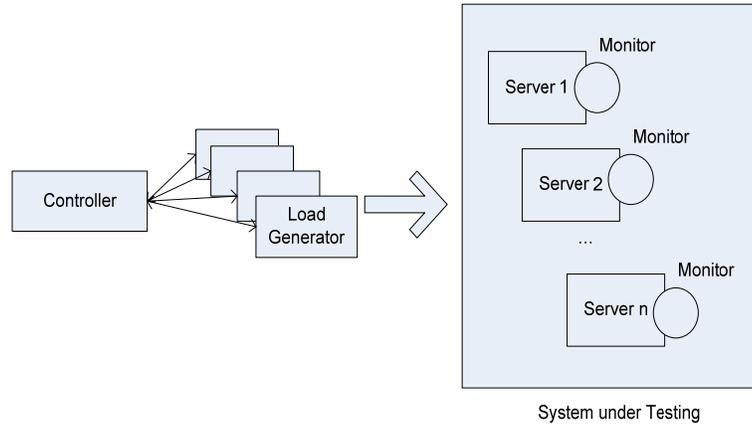


Figure 1: An abstract architecture for performance testing tools

After the performance data are collected, we can analyze the data and calculate performance measurements. Performance testing tools usually provide analysis and reporting tools. The system complies with the performance requirements if the performance measurements meet the expectations stated in the requirements specification.

Average workloads and peak workloads are especially important for software performance testing on concurrent systems [20]. Performance testing with average workloads shows how the system performs under regular usage from the users' perspective. Performance testing with peak workloads provides information about performance degradation under heavy usage. For a software system, operational profiles can be used as average workloads [2]. An operational profile of a software system is a complete set of the operations the system performs, with the occurrence rates of the operations [14]. The occurrence rates can be collected from the field usage, or obtained from existing business data, or from the information of a previous version or similar systems [15].

In addition to a load generator and monitor, a profiler is another often-used performance testing tool. A profiler is a type of performance monitor that provides code-level measurement, including timing, memory usage, and so on [8]. Profilers are useful when we want to locate performance bottlenecks in a software system. A performance *bottleneck* is the location in software or hardware where the performance is lower than that in other parts of the system and thereby limits the overall throughput. To demonstrate how to locate performance bottlenecks with a profiler, consider the Java code in Figure 2.

Performance Testing

```
private static String reverse(String s) {
    if(s.equals("")) return "";
    String temp = s;
    String result = "";
    for(int x=0; x<s.length(); x++) {
        result += temp.substring(temp.length() - 1);
        temp = temp.substring(0, temp.length() - 1);
    }
    return result;
}
```

Figure 2: The reverse method

The code shows a bad practice in Java programming: using “+” to concatenate strings. Figure 3 shows the result generated from a profiler⁴. The result shows that, in the reverse method, a large amount of time is spent on `StringBuilder.append` and `StringBuilder.toString`. These two methods are called if “+” is used for string concatenation.

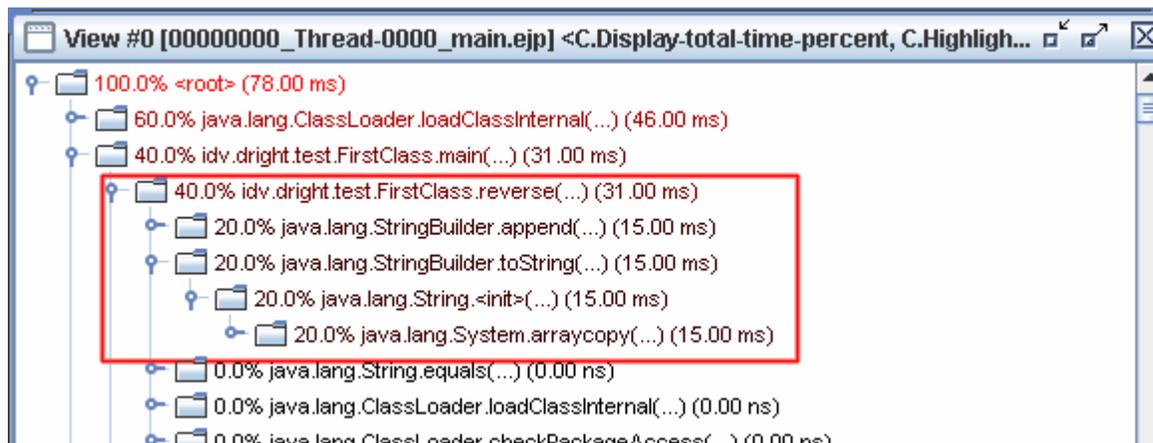


Figure 3: Profiling result for the reverse method

A better way to concatenate two strings is via a `StringBuffer`. Figure 4 shows the modified reverse method.

```
private static String reverse(String s) {
    if(s.equals("")) return "";
    String temp = s;
    StringBuffer result = new StringBuffer();
    for(int x=0; x<s.length(); x++) {
        result.append(temp.substring(temp.length() - 1));
        temp = temp.substring(0, temp.length() - 1);
    }
    return result.toString();
}
```

Figure 4: Modified version of the reverse method

⁴ The profiler used here is Extensible Java Profiler. See <http://ejp.sourceforge.net/> for more details.

Performance Testing

Figure 5 shows the profiling result after the modification. We can see that the new reverse method takes virtually no time to complete.

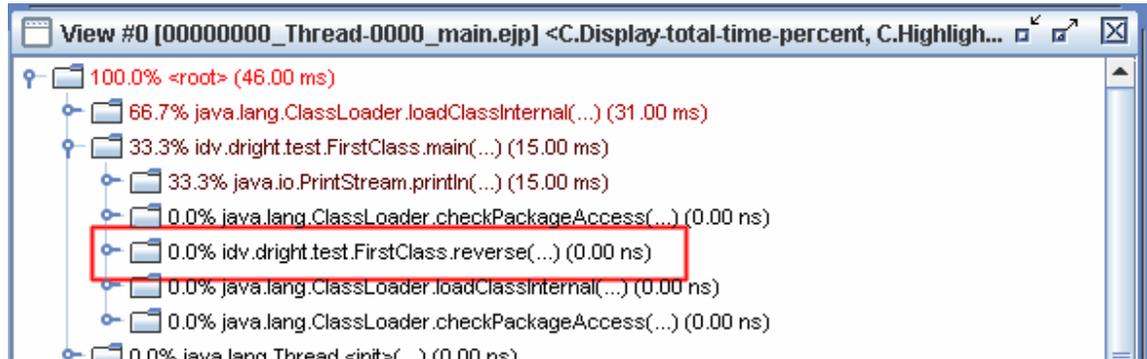


Figure5: Profiling result for the modified reverse method

Benchmarks are also used as an indication of software performance. A *benchmark* is a *standard against which measurements or comparisons can be made* [12]. If the test results meet or exceed the benchmarks, the system is said to be performing well. The goal of the benchmarking technique is to test how the system works when deployed in real life environment. However, testers should ensure that the validity of the benchmark is assessed periodically to prevent measurement against an obsolete benchmark.

A clearly defined set of expectations is essential for meaningful performance testing. The definition of metrics to assess the comprehensiveness of a performance test case selection algorithm relative to a given program [20] are essential to definitively determining if performance requirements have been met. Two main variables are considered: load and response time:

- *Expected load in terms of concurrent users*: Test cases can be written to simulate the users at the client side. The number of simultaneous users can be increased gradually at the client side till the system crashes in a form of testing called stress testing. *Stress testing is testing conducted to evaluate a system or component at or beyond the limits of its specified requirements* [12]. When the system crashes, the operational limits of the system can be ascertained.
- *Acceptable response time*: The delay time between the request sent by the user from the client and the response from the server side should have an upper bound value. If the request does not come back within the specified limit, we can conclude that there is a performance issue that is to be addressed. For example, whenever a patient tries to login from his browser by providing his Patient ID and password, he gets validated from the server. The patient will be very frustrated if he or she is kept waiting for what is perceived as an excessive amount of time.

3. When to Consider Performance in Development Lifecycle

To build a software system with acceptable performance, the development team needs to take performance into consideration through the whole development cycle [9]. During requirements specification and analysis, performance requirements need to be specified. Although the design and implementation details are not usually available during the requirements phase, the performance requirements should still capture the desired performance level. At design and architecture stages, performance models can provide early feedback on the performance of the design. A performance model is a model that is used to analyze the performance of a system. After analyzing the performance of several design candidates, the development team can select a design that can achieve the desired performance level. If a requirement specifies unreasonable performance expectation, the development team can identify the problem before the software is implemented, and negotiate with the customer for an achievable performance requirement. At development phase, the developers implement the software system based on the design that is validated with performance models. During software testing, performance test cases are instrumented to make sure that the performance of the resulting system is at least as good as specified in the requirements.

Software performance engineering (SPE) is a systematic, quantitative approach to constructing software systems that meet performance objectives. SPE is an approach to integrate performance engineering into software development process [23]. In SPE, performance models are developed early in the software lifecycle, usually at the architecture design stage, to estimate the performance and to identify potential performance problems. SPE prescribes principles for creating responsive software, the data required for evaluation, procedures for obtaining performance specifications, and guidelines for the types of evaluation to be conducted at each development stage. SPE uses adaptive strategies, such as upper- and lower-bounds estimates and best- and worst-case analysis to manage uncertainty. For example, when there is high uncertainty about resource requirements, analysts use estimates of the upper and lower bounds of these quantities [23].

Consideration of performance requirements is crucial in the early development stages, when important architectural choices are made [7]. When performance is not considered early in the process, the development team may end up with “universally slow code,” meaning that, even with a profiler, bottlenecks cannot be identified because every piece of code is slow. In this situation, “*tuning*” code to improve performance is likely to disrupt the original architecture, negating many of the benefits for which the architecture was selected. *Performance tuning is the process of transforming code that does not meet the performance requirements into code that meets the expected performance level without changing the behavior of the code* [18]. Additionally, “tuned” code may never have the performance of code that has been engineered for performance [23]. In the worst case, the system design cannot be tuned to meet performance goals, necessitating a complete redesign or even cancellation of the project [23].

Early testing of software systems helps to reveal the modifications like design changes, hardware compatibility issues, which might be required in the product. Changes can be

made more easily and cheaper if faults, including those that drive performance problems, are detected as early as possible [5]. Estimates of performance are used to reveal flaws in the original architecture or to compare different architectures and architectural choices. Development of models which give performance estimates early enough during the development may give useful hints of the performance and help identify bottlenecks [7].

Another camp claims that the developers should not optimize the system performance until the functionalities are implemented. Auer and Beck list a family of software efficiency patterns called *Lazy Optimization* [1], which reflects the famous quote from Knuth that “Premature optimization is the root of all evil in programming,”⁵ and the “You Aren’t Gonna Need It (YAGNI)” philosophy of Extreme Programming [4]. These patterns can be summarized as follows. Early in development, the system performance is estimated with a short performance assessment. Rough performance criteria are specified to show performance concerns in the system, and are evolved as the system matures. Tune performance only when the functionality works but does not pass the performance criteria. Smith and Williams, authors of SPE, criticize that the “fix-it-later” attitude is one of the causes of performance failures [17].

4. Conclusion

Many project teams expend a great deal of resources testing the functionality of the system, but spend little or no time doing performance testing, even though performance problems often significantly impact the project’s ultimate success or failure [17]. A performance test suite should include test cases that are a representative portrayal of the workload of the system under test, as well as a representative portrayal of the workload that is likely to be resident with the system when it is operational.

5. Acknowledgements

Praveen Yeri contributed to an early draft of this introduction.

Table 1: Key Ideas for Performance Testing

	Performance is highly visible to the software user. A system that runs too slow is likely to be rejected by all users.
	Performance can easily be overlooked when customers and requirements elicitors focus too much on the functionality.
	Often the performance of tuned code is not as good as the performance of code that has been engineered for performance.
	A clearly defined set of expectations is essential for meaningful performance testing.
	The performance issues that arise may be because of one or some of the reasons: lack of performance estimates, the failure to have proposed plans for data collection, or the lack of a performance budget.

⁵ “Computer Programming as an Art,” 1974 Turing Award lecture.

Glossary of Chapter Terms

Term	Definition	Source
Benchmark	(1) A standard against which measurements or comparisons can be made; (2) a procedure problem, or test that can be used to compare systems or components to each other or to a standard as in (1).	[12]
bottleneck	the location in software or hardware where the performance is lower than that in other parts of the system and thereby limits the overall throughput	
black box testing (also called functionality testing)	Testing that ignores the internal mechanism of a system or component and focuses solely on the outputs generated in response to selected inputs and execution conditions.	[12]
latency	The time interval between the instant at which an instruction control unit issues a call for data and the instant at which the transfer of data has started; the delay between request and completion of an operation	[12] [7]
load driver	a software program that takes the workload information as input, and generates requests that mimic the user behavior.	[18]
monitor	A program that observes, supervises, or controls the activities of other programs.	[21]
performance	Degree to which a system or component accomplishes its designated functions within given constraints, such as speed, accuracy, or memory usage	[12]
performance requirement	Requirement that imposes conditions on a functional requirement; for example a requirements that specifies the speed, accuracy, or memory usage with which a given functionality must be performed	[12]
performance testing	Testing conducted to evaluate the compliance of a system or component with specified performance requirements	[12]
profiler	a type of performance monitor that provides code-level measurement, including timing, memory usage, and so on	[18]
resource consumption	the amount of memory or disk space consumed by the application	[16]
stress testing	testing conducted to evaluate a system or component at or beyond the limits of its specified requirements	[12]
throughput	The amount of work that can be performed by a computer system or component in a given period of time; for example the number of jobs in a day	[12]
tuning	the process of transforming code that does not meet the performance requirements into code that meets the expected performance level without changing the	[18]

	behavior of the code.	
white box testing	Testing that takes into account the internal mechanism of a system or component	[12]

REFERENCES:

- [1] A. Avritzer, J. Kondek, D. Liu, and E. J. Weyuker, "Software Performance Testing Based on Workload Characterization," the 3rd International Workshop on Software and Performance, Rome, Italy, 2002, pp. 17-24.
- [2] A. Avritzer and E. J. Weyuker, "Generating Test Suites for Software Load Testing," International Symposium on Software Testing and Analysis, Seattle, WA, 1994, pp. 44-57.
- [3] V. R. Basili and J. D. Musa, "The Future Engineering of Software: A Management Perspective," in *IEEE Computer*, vol. 24, 1991, pp. 90-96.
- [4] K. Beck, *Extreme Programming Explained: Embrace Change*, Second ed. Reading, Mass.: Addison-Wesley, 2005.
- [5] B. W. Boehm, *Software Engineering Economics*. Englewood Cliffs, NJ: Prentice-Hall, Inc., 1981.
- [6] E. J. Braude, *Software Engineering: An Object-oriented Perspective*. New York: John Wiley and Sons, Inc., 2001.
- [7] G. Denaro, A. Polini, and W. Emmerich, "Early Performance Testing of Distributed Software Applications," Workshop on Software and Performance, Redwood Shores, California, 2004, pp. 94 - 103
- [8] D. G. Firesmith and B. Henderson-Sellers, *The OPEN Process Framework: An Introduction*: Addison-Wesley, 2002.
- [9] G. Fox, "Performance Engineering as a Part of the Development Life Cycle for Large-Scale Software Systems," The 11th International Conference on Software Engineering, Nice, France, 1990, pp. 52-62.
- [10] C. Ghezzi, M. Jazayeri, and D. Mandrioli, *Fundamentals of Software Engineering*. Englewood Cliffs, NJ: Prentice Hall, 1991.
- [11] C.-w. Ho, M. J. Johnson, L. Williams, and E. M. Maximilien, "On Agile Performance Requirements Specification and Testing," Agile 2006, Minneapolis, MN, 2006, pp. 47-52.
- [12] IEEE, "IEEE Standard 610.12-1990, IEEE Standard Glossary of Software Engineering Terminology," 1990.
- [13] G. Kotonya and I. Sommerville, *Requirements Engineering: Processes and Techniques*. Chichester: John Wiley and Sons, 1998.
- [14] J. D. Musa, "Operational profiles in Software-Reliability Engineering," *IEEE Software*, vol. 10, no. 2, 1993, pp. 14-32.
- [15] J. D. Musa, "Chapter 2: Implementing Operational Profiles," in *Software Reliability Engineering: More Reliable Software Faster and Cheaper*, 2nd ed. Bloomington, IN: AuthorHouse, 2004, pp. 93-151.
- [16] B. A. Nixon, "Managing Performance Requirements for Information Systems," the 1st International Workshop on Software and Performance, Santa Fe, NM, 1998, pp. 131-144.

Performance Testing

- [17] R. Pressman, *Software Engineering: A Practitioner's Approach*, Fifth ed. Boston: McGraw Hill, 2001.
- [18] C. U. Smith and L. G. Williams, *Performance Solutions: A Practical Guide to Creating Responsive, Scalable Software*. Boston, MA: Addison-Wesley, 2002.
- [19] F. I. Vokolos and E. J. Weyuker, "Performance Testing of Software Systems," 1st International Workshop on Software and Performance, Santa Fe, NM, 1998, pp. 80-87.
- [20] E. J. Weyuker and F. I. Vokolos, "Experience with Performance Testing of Software Systems: Issues, an Approach, and Case Study " *IEEE Transactions on Software Engineering*, vol. 26, no. 12, December 2000, pp. 1147-1156.
- [21] Wikipedia, "<http://www.wikipedia.org/>," no.
- [22] L. G. Williams and C. U. Smith, "Five Steps to Solving Software Performance Problems," <http://www.perfeng.com> no., June 2002.
- [23] L. G. Williams and C. U. Smith, "Performance Evaluation of Software Architectures," The 1st international workshop on Software and performance, Santa Fe, NM, October 1998, pp. 164-177.