

Preamble

This material is released under a Creative Commons License agreement <http://creativecommons.org/licenses/by/2.0/uk/>. In informal terms we, the authors, give you permission to do most things with the material provided you:

- a) Acknowledge us as the authors of the original material, and retain the acknowledgement in copy of the material, and in any of your material based on ours,
- b) Do not pretend you wrote it,
- c) Do identify any changes you make as your work, not ours.

The material includes opinions of the authors, and may be different from work published by others. We have quoted and referenced work from third parties, and in a few cases have knowingly used or adapted ideas or content from third parties. Where appropriate we ask for permission from the third parties.

However we make mistakes, as most people do, and may have made mistakes in this work. If you spot something you feel is wrong, or simply something you feel could be improved please let us know via this email address nft.introduction@commercetest.com

We are Stuart Reid and Julian Harty, and are the authors of this material: **An introductory course on non-functional software testing**. If you wish to create derived works you need include the following acknowledgement: at the start and end of the derived works: **This work is based on original material by Stuart Reid and Julian Harty**. We like to know whether our work is being used so please tell us how you're using the content.

Finally, this work is still in a draft form so you may find inconsistencies, gaps, incomplete content, etc. We hope you will find the content useful anyway and we are very happy to receive suggestions for improvement to the email address mentioned above.

Software Performance

Let's start by looking at some definitions of performance and performance testing. The following are all taken from IEEE 610.12, which defines many terms. Note: the material is from 1990

Definition of Performance

- The degree to which a system or component accomplishes its designated functions within given constraints, such as speed, accuracy, or memory usage.
IEEE 610.12:1990

Definition of Performance Testing

- *Testing* conducted to evaluate the compliance of a system or *component* with specified performance requirements.
IEEE 610.12:1990, BS-7925-1:1998

Definition of Performance Requirements

- A requirement that imposes conditions on a functional requirement; for example a requirements that specifies the speed, accuracy, or memory usage with which a given function must be performed.
IEEE 610.12:1990

Using the definitions above, the performance of software is concerned with its behaviour in terms of:

- The timeliness of its responses, and the throughput of work: how much work can the software perform in a given period?
- What resources the software needs in order to perform its tasks. For instance there may be a trade-off between response time and the amount of memory used by the software to cache data.
- What accuracy is required by the user e.g. in the results returned by a search algorithm?

These definitions do not mention things like volumes of activity, the impact of concurrent usage, etc. However these topics are relevant for performance testing today.

Here is an informal description of performance testing:

Performance testing helps to determine whether software will meet the performance requirements such as response times, concurrency, resource consumption and volumetrics.

Performance testing touches on many areas including:

- **Load testing**
- **Stress testing**
- **Volume testing**
- **Soak testing**
- **Scalability**

Performance attributes interact with many other non-functional aspects ranging from **security** to **usability**.

Technical views of performance testing

There are a number of technical views of performance testing, where the focus is on performance from a single perspective. These views are:

- Isolated, single user performance
- Aggregated, multi-user performance
- Deviation, best, average and worst case
- Scaling and capacity planning
- Queue management
- Concurrency issues

Isolated, single user performance

Here performance is considered and measured from a single user's viewpoint i.e. "How does the system perform for *me*?" Performance for other users is not relevant: unless it impacts on the performance perceived by this, isolated user.

Things to consider include:

- The impact of any network connection used to support the user's requests for service, especially if slow links such as dial-up connections over modems are used
- The size and numbers of requests and responses related to this user
- The specification of the user's individual computer. Performance of the user's computer can have a significant impact on the performance experienced by the user.
- The characteristics and behaviour of the user e.g. are they 'happy-clickers' who incessantly click on the refresh button while waiting for a response? These clicks can cause many requests to be cancelled and reissued, which increases both the load on the system and the overall response time.

Aggregated, multi-user performance

Aggregated performance measures the overall performance experienced by multiple users e.g. "How does the system perform for *them*?" Performance for individual users is not particularly important, except in so far as it affects the aggregate values.

Load testing is a good example where the tester is interested in the aggregate performance e.g. on the impact of having more users using a shared system at the same time. In an ideal world response times would not deteriorate, and tasks would take the same time to process. However in practice response times do deteriorate and tasks can end up taking much, much longer than expected.

Sometimes we can identify distinct groups of users e.g. some may have older computers, be located at another site, or they may perform different activities using the software. If so, and if the grouping is sufficiently different from the norm, the performance testing generally needs to reflect these groupings. For example we might want a test that includes, or simulates, users connecting from a remote location.

Deviation, best, average and worst case

The performance of any system may increase or decrease from one result to the next for a variety of reasons e.g. the effect of other load on the system. Therefore, the requirements may need to state an acceptable variance from an average, including 'worst-case' acceptability. Experienced performance testers often use measures such as the mean, and 95% percentile to represent the average and 'worst-case' respectively. Note: this course does not include the reasons behind these measures as they take some time to explain sufficiently.

Sometimes we can accept large variances provided the mean (average) is acceptable, however we may need the performance to be more consistent e.g.:

- Results from a search engine: these may be acceptable even if one request is serviced in 0.1 seconds and another in 3 seconds provided the mean is less than 2 seconds.
- In a real-time system such as an electronic pacemaker embedded into a person's heart, greater consistency and a much lower variance are required.

Statistical calculations are often used to measuring deviation, best, average and worst case results.

Scaling and capacity planning

Scalability and capacity planning are related activities: scalability considers how well software can scale to cope as volumes and/or load increases, while capacity planning manages the system to ensure there is sufficient capacity to cope with the increases of load and/or volume.

Note: capacity planning, and to a less extent scalability, may need to address *decreases* in load and/or volume.

Queue management

Many systems include queues as a way of coping when demand exceeds the ability of the system to process the demand immediately.

Concurrency issues

When multiple tasks run in parallel, they run concurrently. If the tasks need to share resources, and they often do, there may be conflicts, such as:

- Starvation where one task 'starves' another, e.g. by blocking other tasks from accessing a resource such as a common file.
- Slow downs, when access to shared resources is controlled e.g. to limit access to a single task at once
- Corruption, where updates from several tasks are not synchronised properly

What is unique about performance testing?

Performance testing measures numbers e.g. response time, CPU usage, number of concurrent users, etc. It also helps to find concurrency problems.

Performance is one of the core, often assumed, requirements of stakeholders for a given application or system. The stakeholders implicitly want the performance to be adequate and are concerned when the system fails to meet their assumptions or requirements.

Comparing Performance testing with...

Functional testing

Performance testing is different from functional testing as performance testing builds on functional tests by adding additional: requirements to, constraints for, or combinations of test cases. Often only a subset of relatively straightforward functional test cases are selected to become performance test cases, for the following reasons:

- Complex test cases are harder to augment with performance requirements,
- Generally the performance testing models the most common activities and tasks first, these tend to be captured in the more straightforward test cases.

Security testing

Performance testing can be used to test **denial-of-service** i.e. where the test tries to deny service to legitimate users of the system, often by subjecting the system to large volumes of requests.

Both performance and security testing rely on the testers having technical skills, especially when automating the test execution.

Skills, such as being able to analyse and **reverse-engineer** protocols and interfaces are common to both security and performance testing.

Performance testing measures numbers, etc. while security testing assesses confidentiality, integrity and availability of the software. Security testing includes threat modelling, attacking a system with the intention of finding vulnerabilities and exploiting those vulnerabilities, etc.

Usability testing

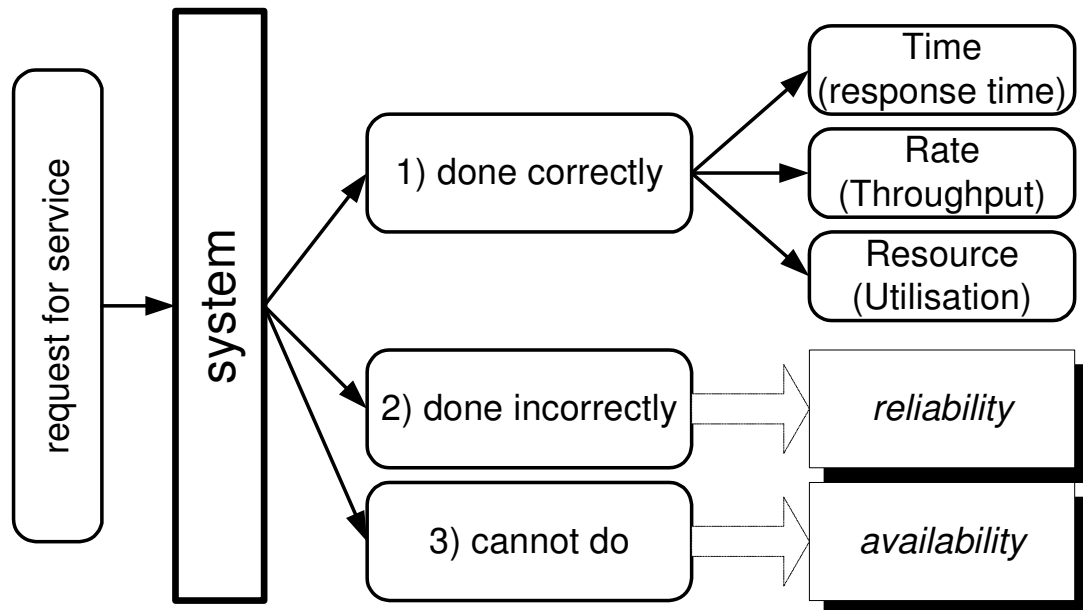
Performance is one aspect of usability, and one measure of usability is how easily and quickly a user can perform their tasks. [Link to Jakob Nielsen 2001a]. Usability testing often tests the user-interface and is affected by the users who use, or are expected to use the system.

Usability testing tends to focus on the human-computer interface and be less technical in nature.

Availability and Reliability

When measuring performance we may encounter one of three results for each request. The following diagram shows the three possible outcomes for a request for service:

1. The request was successfully processed without error
2. The request was processed, but completed incorrectly (with or without an error being reported)
3. The request was rejected



Based on Figure 3.1 page 33 of "The Art of Computer Systems Performance Analysis" Raj Jain

When a request is processed correctly we want to record the results relevant to the performance e.g. response time.

For 'done incorrectly' we should record the types of error, whether there is any pattern to their incidence, and how often they occur. These failures affect the reliability of the software and can be measured and assessed as part of the reliability testing work.

For 'cannot do' we need to know the time interval between 'cannot do's and how long the system is unavailable. These failures affect the availability of the software and can be measured and assessed as part of the availability testing work.

Special stakeholders

The performance of software and system is important to a variety of stakeholders, including:

- End users,
- Customers, including the investor or project sponsor who needs a return on the investment, or outlay, required to commission the software / system,
- The Operations team who need to manage the software in service e.g. in terms of capacity planning, etc.

Benefits and Risks associated with performance testing

The importance of performance testing

Without performance goals or requirements the software is likely to disappoint one or more stakeholders e.g. if the response times are too slow, if reports take too long to produce, if the system slows significantly when several users are active concurrently.

Cost of getting it wrong

When a system doesn't perform stakeholders are not happy, and the performance needs to be improved. Improvements generally cost both time and money. Improvements range from reconfiguring a system, which can be relatively inexpensive, up to rewriting most of the software from scratch to fix the underlying architecture or design. Companies have even cancelled projects because the cost of fixing poor performance has been too high to be justified.

Potential conflicts

With functionality

Performance may conflict with functionality for a variety of reasons, e.g.:

- When the project team have to chose between investing their limited resources on improving performance versus adding new functionality
- Where richer functionality costs more in terms of resource-consumption e.g. where a more impressive user-interface requires lots more code, memory and CPU to implement.

However, functionality and performance need not conflict, when performance requirements are considered earlier in the software development lifecycle. The code can then be developed to meet both performance and functional requirements.

With other attributes

Again, performance has the potential to conflict with many other non-functional requirements e.g.:

- Security: Some security mechanisms require lots of computing resources e.g. SSL for web pages and adversely affect performance unless the designers use things like dedicated hardware accelerators to implement the SSL support. Also, increasing the data validation is likely to consume more resources than assuming that all inputs are correct.
- Usability: A richer user-interface may be more 'usable' yet require more computing resources to deliver it.
- Robustness: robust software generally has more error-checking code, again this often consumes more resources than assuming inputs are ok and error-checking is likely to increase the processing time required for each task.

Good architects, designers, and coders will consider the various, potentially conflicting non-functional requirements, and do what they can to align and incorporate these various non-functional requirements so they do not conflict with each other.

Examples of specifying performance requirements

Performance requirements can be hard to specify precisely, and although people often try to specify them in absolute terms e.g. “all requests must complete in less than 1 second”, they end up compromising on the requirements when a system fails to meet them.

Good performance requirements tend to be practical, and to use statistical measures e.g. 95% of searches will complete in less than 2 seconds, with a mean of no more than 1.5 seconds. The accuracy of the search must exceed 98% as ranked by the users¹.

Such a requirement allows for a small percentage of searches taking longer than 2 seconds, e.g. some may even exceed 5 seconds. This may be acceptable provided the vast majority of users receive accurate results within 2 seconds.

Performance requirements on a shared system should be in context e.g. they might need to take into account other activities on the system such as database backups, the physical hardware, other user activities, etc.

ID	NFR.25
Description	Response time for a simple-search. The response time when users use the simple-search feature to search for a single word or a text phrase. Searches that include regular expressions are excluded from simple-searches and are covered in the Advanced-search requirement.
User Requirement	UR.4
Primary Attribute	Performance ² (interactive).
Measure	Interactive response in tenths of a second.
Basic Requirement	Mean <= 1.5 seconds. 95% <= 2.0 seconds.
Ideal Requirement	Mean <=1.0 second. 95% <= 1.5 seconds.
Current value	Mean = 2.2 seconds. 95% = 6.5 seconds.
Technique	Time each search. Note: details of the test design aren't included in this course.
Test Requirement	Timing device (e.g. a stopwatch).

¹ Some form of measure of the accuracy would need to be agreed and defined to enable the testers to measure the accuracy e.g. by using a **test oracle** as a reference. Alternatively if the intended users are involved in the testing they may have sufficient domain knowledge to validate the results.

² Could also be classed as Usability.

	Test data and test cases will include the 'Top50' searches from the current system, and the 'SpecialSearchCases', documented in the project folder.
Operational Requirement	Field tests, and analysis of timing records generated by each search-engine server.
Conflicts	None.
Priority	7 (on range 1-10 for this project).

Approaches for testing performance

A diligent tester can often perform single-user performance tests, providing the requirements are reasonably clear and the tester has a way to measure the performance data e.g. by using a stopwatch to measure response times.

When the test execution is automated, the tester needs to be able to use the tool competently. Automated tools tend to generate large volumes of data that needs to be sifted and interpreted. Tools do not extrapolate the results or compare them with user-oriented performance requirements, so the tester may need to interpret and extrapolate the results and decide whether the expected performance requirements were met.

When is performance testing performed in the SDLC?

Performance testing can start as soon as code has been developed, and the results may help the developers to find and fix performance issues at a component level. However the majority of performance testing takes place during component integration testing, system testing, and may continue through intersystem integration testing and acceptance testing until the software is commissioned in a live environment.

Earlier in the SDLC performance modelling and simulation help to identify performance issues. And once the software is in use, performance metrics can be captured and analysed to find out whether the software is meeting its performance requirements.

Where is performance testing performed?

As the physical environment often affects the performance e.g. the specification of the servers, network and client computers, performance testing is ideally performed in a similar environment to that intended to be used when the software is in production. This reduces the chances of the results misleading the testers, as testing in dissimilar environments requires the results to be extrapolated and the extrapolation process is not scientific, or very accurate.

What special resources are required?

Performance testing often needs automated load generation tools, especially to simulate multiple users, load, volumes of transactions, etc. Testers need to understand and use software that monitors resource utilisation; these software tools include things like:

	Operating System family	
Resource monitored	Windows NT	Unix, Solaris, Linux
CPU	Task Manager Perfmon	pstat top
Network	Netmon Ethereal Commercial tools	Ethereal Commercial tools
Memory	Task Manager Perfmon	vmstat
Disk	Perfmon (with disk	vmstat

	counters enabled)	du df
--	-------------------	----------

Note: these tools are well recognised and established. You may also have more powerful tools available. If so, and if they offer significant benefits, why not use them?

The environment should be similar to the expected production environment where feasible. Where differences exist, the effect of the differences need to be considered and accounted for when analysing the results and reporting on the findings.

Costs of performance testing

Performance testing can be inexpensive, e.g. if the testing is relatively simple and the performance requirements are easy to measure without commercial software tools. In practice, performance testers often use high-end commercial performance testing tools, which tend to be expensive to license. Inexpensive and tools that are free to use e.g. from open source projects are available and may be more than adequate once the tester has become familiar with the tool. For commercial tools, testers may benefit from commercial, and therefore relatively expensive training courses.

The test environment may be expensive to build, configure, and maintain. Costs include preparing test data, especially when large volumes of representative data are required.

Finally, relatively few companies employ full time performance-testing experts, so when such expertise is required companies need to account for the cost associated with employing performance-testing experts.

Simple example of performance test techniques

TURA

TURA

Starts by adding additional requirements and conditions to existing functional test cases

- Time
- Users
- Resources
- Accuracy

$X + Y = Z$ in time T , with U users, and R resources, to A accuracy

TURA is a simple and effective technique used to create performance test cases. It adds additional constraints to existing functional test cases e.g. a simple-search must complete within 1.5 seconds (Time), with ≤ 50 users performing simple-searches concurrently (Users), where each search process consumes less than 1MB RAM (Resources), and the results are at least 80% accurate (Accuracy).

Traversal and Interactivity

Traversal and Interactivity

Models typical behaviour of users:

- How they use the system
- Interaction between users
- e.g. buyers and sellers
- Frequency of use
- Variations in behaviour

Generally converted into test scripts for automated tools, although large teams can act out the roles directly.

Traversal and Interactivity is a testing technique that aims to emulate real-life usage as closely as possible.

Start by describing ways that users will interact with the system. Find out how often they perform each activity and whether it depends on other activities – either of themselves or other users of the system. Convert each activity into a test script. Provide a way to vary the details of each activity. Sources of information include:

- UML 'Use Case' diagrams
- Logs from similar systems e.g. the can be extracted from web server logs
- Interviews with stakeholders e.g. with the marketing department to find out projected usage once the software has been deployed.

Markov models can model the choices made by the users, and Operational Profiles model the overall volume of demands placed on the system.

Operational Profiles are introduced in the Dependability section of this material.

Use of statistical analysis methods such as Poisson and Erlang can help to predict and schedule the distribution and frequency of requests.

To ensure that we are not simply testing the caching capabilities of the system there needs to be some variation in the behaviour, e.g. in the size and contents of a request.

Scenarios and Workloads

Load on a system is seldom uniform. The variations in load means the demand on the system will also vary.

Scenarios and Workloads (similar to operational profiles) are used to exercise the software in order to find out whether the software meets its performance requirements as the load varies.

Scenarios are a predicted set of circumstances, and/or a predicted sequence of events. For example a month-end scenario may describe the circumstances for a month-end e.g. the first working day of the following month, and the sequence of events that would be expected to occur e.g. balance customer accounts, check stock levels, reset sales targets, produce monthly reports, etc.

Workloads describe the amounts, and types, of work to be done, especially in a specified period by a person, machine, etc.

Performance test cases that include scenarios and workloads tend to include various more detailed test cases e.g. 'balance customer accounts', however the overall requirement, and therefore the focus of the test is on the overall performance and behaviour of the system, rather than the performance of each individual test case.

References

<http://www.perftestplus.com> - A great resource on performance testing

The art of computer systems performance analysis

Raj Jain,

ISBN 0-471-50336-3, Wiley, © 1991

Capacity Planning for web performance

Daniel A. Menascé and Virgilio A. F. Almeida

ISBN 0-13-693822-1, Prentice Hall, © 1998

Gain econfidence & load testing for eConfidence

Privately published by Segue.com

The Web Testing Handbook

Steve Splaine and Stefal P. Jaskiel

ISBN 0-9704363-0-0, STQE Publishing, © 2001

-- End of document --