A Beginners Guide To Performance Testing

Stephen Jenkins

 Make Note  Avoid Doing  Top Tip

Abstract

You would be surprised at how many companies do not understand the significant benefits performance testing can bring to their organisation by:

- Leaving performance testing to the end of their development effort.
- Do no performance testing at all in a bid to meet artificial timelines.
- Over-engineer it to the point that the results are useless and ignored.

The reason for this is that there is a perception that it is difficult to do, which is of course nonsense.

There are many misconceptions surrounding performance testing, such as:

- Outsourcing is the best and most cost effective approach to performance testing which is not true.
- That keeping the testing in-house requires the skills that only highly paid consultants possess which again is not true.
- It takes a long time to build and execute and will therefore impact on the delivery timescales which is again untrue.

Even if any of this was true a small slippage on delivery should be more than palatable if the end result gives you an application that performs under load.

In this white paper we are going to demonstrate that with a little thought and a bit of common sense any company can execute performance testing regardless of how complex the application under test.

Introduction

Performance testing is not difficult given the right approach and an understanding of what you are really trying to achieve.

Common pitfalls include, but are not limited to as there are many more, too numerous to mention:

- Throwing artificial load at systems with little thought for its accuracy.
- Trying to performance test the whole system.
- Getting caught out with legacy architecture.
- Not including your business owners in the building of requirements.
- Not including technical resources (Tech Leads, Architects) in the building of requirements.
- Not building performance alongside development and seeing it as a before we go-live activity.
- Overcomplicating tests.

With a little guidance performance testing can become a relatively straightforward part of your QA journey and one that ensures that your users experience is not damaged by a poorly performing application.

This white-paper looks to expose the many myths surrounding performance testing and provides a relatively straightforward, in places controversial and effective approach to the discipline.

What is Performance Testing?

There are many definitions, all different, all correct because the definition of performance is what it means to your company and your applications under test.

There are some basic guidelines and types of testing but the way it is executed and how it is executed differs, this is an important concept in your journey towards successful performance testing.

Define what performance testing means in terms of your application and organisation and do not try and replicate what other organisations do or encourage you to do.

Ok we understand that you may not have the capability to run your own performance testing and you wish to outsource but you should really try and keep the capability in house and use your own resources where possible, even if they are contractors.

If outsourcing is your only visible option then only outsource the execution not the definition of requirements and definition of approach this needs to be owned by you.

The first step on this journey to defining your own performance testing is getting your requirements right and we will discuss this next.

It is important to understand that performance testing it is not is just throwing artificial load and concurrency at a system with no thought as to what you are trying to achieve or how it will make your product better.

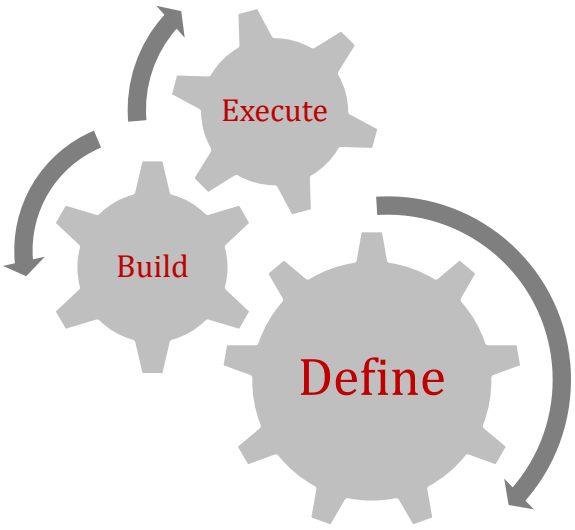
Many guides on performance testing will tell you it encompasses:

- Performance testing,
- Scalability testing,
- Soak Testing,
- Resilience Testing,
- etc.

And while this is true it is not what this white-paper is about.

We are trying to show you how a novice, whether organisation or employee, can start on their journey towards sensible performance testing.

And we will not bombard you with the same old rhetoric about what you should and should not do when it comes to testing, load, volume, concurrency, tools etc.



Test execution and analysis is relatively easy and there are many good resources available.

These resources can do a fantastic job in building comprehensive and leading edge shift-left performance tests as part of agile delivery.

The real skill is defining what your company needs to do to ensure its applications and products perform.

Determining how you translate these test results to something you can measure and these are easily within any companies reach with a little planning and forethought.

Non Functional Requirements and Definition of Scope

This is the start of the performance testing journey; this is where it all begins:

- Not with writing scripts.
- Not with simulating millions of users running complex business journeys.
- Not with writing long winded test plans, which really you should not be doing at all.

There is a good blog post here that covers writing non-functional requirements and it is definitely worth a read

https://octoperf.com/blog/2019/06/26/non-functional-requirements/

In essence you need to define the behaviours of your system, and list them in a concise way under the headings commonly associated with non-functional testing:

- Performance,
- Volume,
- Scalability,
- Maintainability,
- Operability,
- Utilisation,
- Resilience,
- Security,
- Availability,
- Recoverability,
- Disaster Recovery.

Under these categories define how your application needs to behave; let us have a look at a small number of examples just to give you an insight into how we should construct requirements:

You should always use percentiles in your non-functional requirements gathering and testing as you have to be able to exclude anomalies.

Performance

- Web Service A must respond in 500 milliseconds at the 95th percentile
- A logon request to Application A must take no longer than 2000 milliseconds at the 95th percentile

Volume

- Web Service A must be able to support 500 requests per minute

Scalability

- Web Service A must be able to support a transaction rate growth of 10% in line with business growth

Requirements definition is relatively straightforward and this is where you have to start your performance testing journey.

These requirements will provide you with the scope of your testing, the definition of your tests and the way you will measure the results.

No need to generate large amounts of documentation just a set of well-defined requirements.

Do not keep your requirements within the QA function where there they are unavailable to other key project stakeholders.

Get your requirements approved by both business and technical resources, make them available and open to review.

The more the people that contribute the better these requirements become and the better your testing becomes which ultimately improves your product.

This white-paper is titled performance testing I am conscious that we have been discussing

- Performance,
- Scalability,
- Utilisation,
- Volume,
- etc.

Looking at the industry in general most companies look to employ a performance test strategy or performance test engineer or undertake performance testing that encompasses all of these.

To not over-complicate the definition we are calling this paper **A Beginners Guide to Performance Testing** for reasons of simplicity rather than anything else.

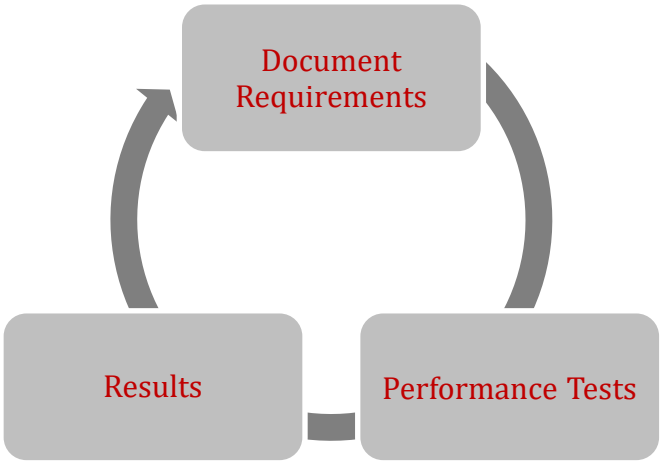
Documentation

This may sound controversial but we do not really need test approaches and test plans.

It is a point for discussion but it is outdated to write to much documentation not to mention the time and effort to get these long winded documents approved.

More often or not when you get to the actual creation and execution of the tests the documentation requires continual, or at best periodic, updating.

Surely we write tests to satisfy our requirements and analyse the results against the same set of requirements, therefore the documenting of requirements should be enough.



Documenting results is important and we will talk about this later, this is one of the most important parts of performance testing.

But writing about what you are going to possibly do that will probably change when you come to do it seems like a waste of time.

Definition of Performance Testing Scope

This is an important concept to consider as you need to be clear about what is in scope for your testing, your requirements are all about the application under test but you need to take into account the wider architectural picture.

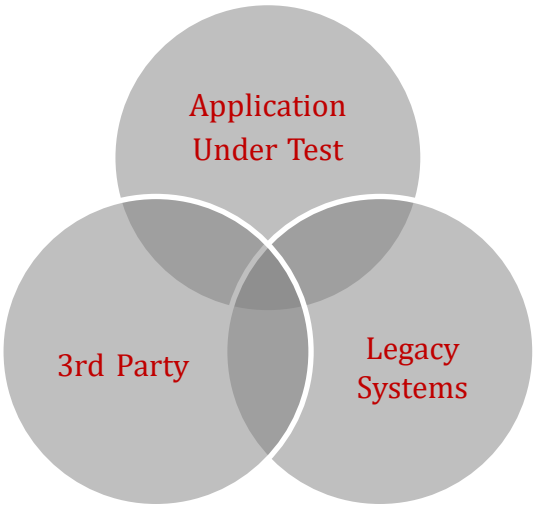
It is uncommon for any new application in the modern world to not be reliant on some legacy technology somewhere, even if it is being phased out, and you need to take this into account.

Now I am conscious that I have spent the last section stating that test plans are out dated and no longer practical and here I am telling you that we need to define our scope somewhere, let us discuss what I mean by scope and then discuss how we document it.

Things to consider when thinking about scope are:

- Are we confining our performance testing to our new application only?
- Do we need to include performance testing on legacy technologies because they will see an increase in load with the introduction of our new system?
- What can we test and what cannot we test, this is hugely important to consider whether it is possible to test some aspects of your new system or your legacy systems?
- Can we engineer a consistent data set or gain access to a legacy system that can handle the load you will place on it during your performance testing?
- Are we testing end to end journeys or are we focussing on individual components of the system in isolation?

You should not worry about moving legacy components or systems out of scope of your testing if it adds to much complexity.



The scope now needs to be incorporated into the requirements document to:

- Ensure we cover legacy architecture.
- Document whether we can test all aspects of the system.
- Define how we will approach our testing.

This effectively concludes the initial and the most important part of performance testing, the definition of what you are going to do.

Even if you feel as an organisation you do not have the capability or skills to execute tests to satisfy your performance requirements you have at least defined what they are and you have something to measure against in the form of a set of documented requirements.

This allows you to hold any third party you may employ to execute the tests for you to account and provide them with a scope for test creation.

The rest of this white-paper will focus on how tests can be easily built to satisfy these requirements and how the outcome of these tests satisfies these requirements should you wish to perform the execution yourself.

It is our opinion, rightly or wrongly that end users, whether internal or external, of your application would overlook a number of functional defects, as long as there was a work around, rather that suffer poor performance

Performance testing is important and getting it right is important and therefore the definition of what it needs to be for your organisation is important.

Invest in YOUR Team

By creating your requirements, you have reached a crossroads in your performance testing journey you now know what you need to do in order to ensure you application performs.

You now need to decide how you go about building and running the testing required to satisfy your requirements and to provide you with confidence in the performance of your application.

As this is a beginners guide and the aim is to demonstrate how straightforward performance testing can be.

The first and the best course of action to achieve this ambition is to invest in your team which may be:

- Up-skilling you current QA team.
- Hiring a performance testing consultant to assist you in getting up to speed.
- Encouraging them to develop their skills as part of their own personal development.
- Work as a team to develop skills through a code-time style forum.

Performance testing is not a one off exercise that once concluded can be ignored.

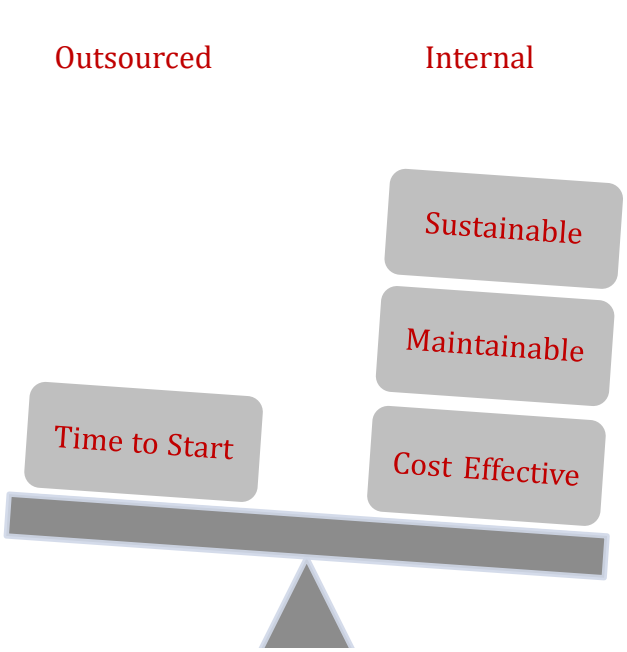
Performance testing needs to be part of your applications regression suite and part of the quality gates for all new releases.

All new applications you develop or existing systems that are being upgraded require a performance test suite to be written so it makes so much more sense to keep the discipline in-house and trust your internal resources.

You could outsource the job to an external consultancy but you run the risk of losing control of the process and being forced to compromise which you should never do when it comes to performance testing.

You would be surprised at how straightforward performance testing can be and how keen your team will be to take on the challenge that you present them.

One of the central aims of this white paper is to give anyone reading it the confidence to start on their performance testing journey knowing that it is something that they can achieve.



- We have shown you how to define your requirements.
- You have encouraged the business and technical resources to buy into your approach.
- We have decided against writing a test approach and test plan and instead have a good set of requirements
- We have wisely decided to invest in your team.

So now you need to decide on a suitable set of tools to help you:

- Build your tests.
- Support your testing framework.
- Define your results analysis process.
- Deliver your development and delivery strategy.

This white paper will not tell you how to build tests and how to define and support a testing framework as this is not the objective of the document and this is primarily due to the fact that it is fairly complex and subjective.

There are many really good articles on the OctoPerf Blog Pages to assist you in this.

https://octoperf.com/blog

Choosing a Tool

You may have already guessed that this paper is very straight to the point and this is not about to change, basically JMeter or LoadRunner when it comes to performance testing tools.

JMeter is an open-source, free to use, tool and LoadRunner is a commercial tool.

There are many performance testing tools on the market but these are the best open-source and commercial tools in terms of sheer volume of knowledge bases and support articles available along with the fact that they have evolved over a number of years and support many technologies.

Now the choice between the two boils down to whether JMeter supports the protocols your applications uses or not.

If it does then it is the best option but if it does not then it is LoadRunner, it is as simple as that.

OctoPerf have a really good article on the side by side comparison of the tools which can be found here.

https://octoperf.com/blog/2019/05/28/jmeter-alternative-loadrunner/

This may seem a bit simplistic but it is very easy to choose your tool, the mistakes come when:

- You start evaluating product after product.
- Running multiple proof of concept sessions.
- Allowing your decision to be swayed by external suppliers.

All this does is make the process of choosing a tool long winded and complicated.

The tool market is an ever changing landscape and new tools arrive that promise you simplicity, ease of use, pain-free, full coverage test scripts in seconds at the press of a button.

This is not true and never will be.

Performance tests that are robust and hardened to ensure they can be easily maintained as your application evolves and can support multiple purposes require hard work and diligence in their creation.

Pick one of the two tools outlined above, depending on your technology, and stick with it.

We have said in the previous section that this document will not expand on how tests are built and frameworks developed, but it is worth understanding modularity and re-usability in test scripting.

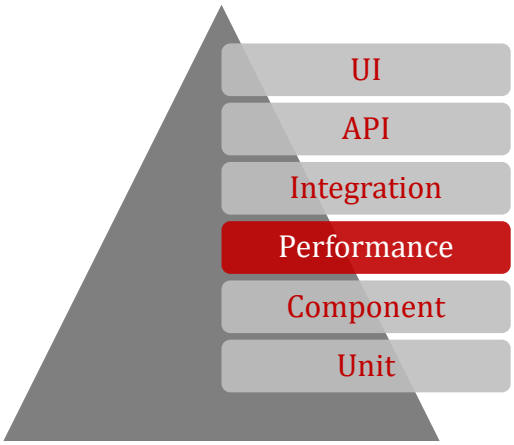
It is worth making sure you do have at least an understanding of the principles of how performance test scripts need to be created so if you do hire a 3rd party to develop them or you do encourage your team to up-skill and learn you will have an appreciation of these principles.

There are many great articles on the OctoPerf blog post site or you can use the internet to gain knowledge with key phrases including:

- Performance test modularisation,
- Data correlation and abstraction,
- Re-usability in performance test scripts,
- Performance testing frameworks and command line interfaces.

Test Execution

When do we run our performance test, at the end or development…no, at the end of each significant release cycle…**no**, just before go-live…**no**, all the time…**yes**.



If we consider our testing pyramid we see performance testing very much at the forefront of the QA effort.

You should not shy away from early and regular execution of your performance testing.

Let us quantify this, a long time ago it was considered normal to execute performance test at the very end of the programme delivery just before go-live.

This was effectively far too late to do anything about poor performance and it soon became apparent the end users was critical of an application that did not respond and it seemed like a good idea to try and execute performance tests earlier.

This concept was met with resistance as it was considered expensive and complex and time-consuming and therefore still remained an afterthought.

Then agile development came along and the QA effort became part of development and delivery running in parallel became normal and it gave a real opportunity to shift our performance testing left which is just a way of saying earlier in the programme lifecycle.

We are conscious that many organisations still use a waterfall approach to technology delivery and development and just because you may not be a fully agile organisation does not mean you should not adopt shift left performance testing.

Agile has just given the performance testing discipline more focus and emphasis and it should be used regardless of your delivery strategy and disciplines used for the development activities.

Now what do we mean by execute all the time?

Our recommendation would be overnight daily when you have more chance of a stable testing platform. By executing overnight, you have a better chance of only the performance test running and no background noise making comparing results more realistic.

The more you execute the more likely you are to spot performance regression and identify an early fix or at least get a tech task visible to the development team backlog.

There is also the additional benefit of regular execution makes sure your tests are up to date and any rework due to an application change can be done straight away.

What does constant execution of performance tests give us, there are many benefits and these include:

- A constant awareness of how your application is performing.
- You can build your tests as your technology stack evolves.
- Make performance improvement recommendations during the development cycles when it is easy to fix or improve.
- Help define hardware requirements for your platform as the testing evolves.
- If performance regresses you can pinpoint to a release, or a code check-in, or a particular component that caused the issue vastly improving and speeding up the root cause analysis.

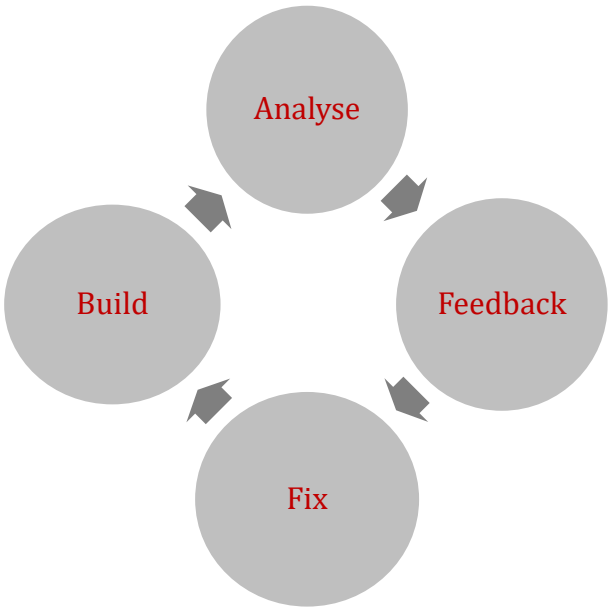
You get the picture, it is just better to run as often as possible, now it is easy to say and the benefits are obvious but the practicalities of doing it can be challenging.

Things to consider are to embed your performance test resources in your development teams so the performance test can be written as functionality is developed.

Make sure your tests specifically meet one or more of your requirements as that is after all the reason you are developing them.

Do not get side-tracked and build tests that do not satisfy any of your requirements as these will need to be maintained and this can take time.

Your daily routine becomes



- Analyse results.
- Feedback to development teams.
- Fix and tests that may have failed because of data or changes to functionality or services.
- Build any new tests and add to your overnight execution schedule.

While this regular execution serves an important purpose and, these have already been discussed, it is important to understand that performance testing does require a formal test cycle.

The formal test cycle is executed to demonstrate to key stakeholders that the system meets your requirements and you can demonstrate this through a set of formal tests and may consist of:

- Peak Volume,
- Soak Test,
- Scalability Test,
- Resilience Test,
- etc.

So what is the point of all this early test execution when you still need to run a formal test cycle to pass you programme quality gate?

- Regular execution of your test will provide a high degree of confidence that these formal tests will pass with no issues found as you will have already found all the major performance issues.
- There will be no script maintenance needed as you will have been constantly maintaining your tests through regular execution.

These are the real benefits of this approach.

The subject of test execution is complex and would require a white paper of its own to cover the topic fully.

This is an introduction to performance testing and therefore all we are saying is do not leave performance testing until the end.

Before we leave test execution, regular overnight tests execution requires some form of execution framework whether this is Jenkins pipelines or CRON scheduled activity.

You need to understand how you will accomplish this but these topics are not suitable for an introduction, we mention this so you are aware and can investigate the subject yourself.

Environments and Scale

On the subject of environments, you need to run your performance tests against an environment that is representative of you production environment as if you do not then you cannot be confident in your results.

Do not extrapolate results gathered against an inferior environment; this is not a good way to measure performance as in all cases performance does not scale linearly.

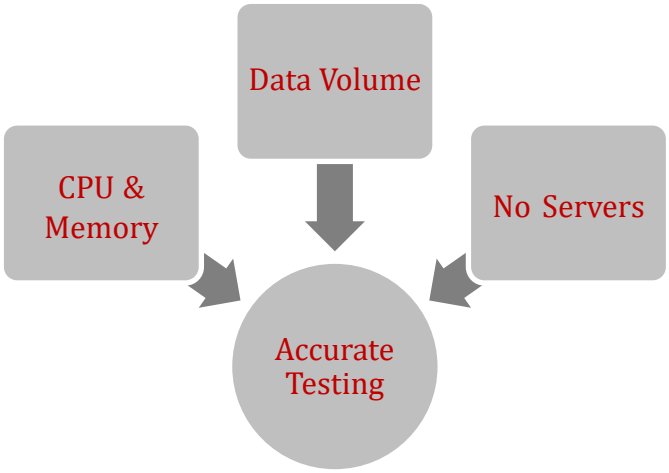
If your production environment has not yet been defined, as you are executing performance testing early in the programme lifecycle, then it must be representative of the expected production environment.

If you can demonstrate that the environment you are testing against does not support your production representative levels of load and concurrency you can recommend that CPU or RAM is increased until it can.

Environment sizing is another benefit of our early adoption of performance testing.

Data is also another important environment consideration, in order to ensure that your database performs, is tuned and that appropriate indexes have been applied, you need representative data volumes in terms of quantity and quality.

Without representative data the performance results generated by your testing will not be representative of performance of your production system.



These environmental considerations can take time to put in place and by addressing it early is another positive from early performance testing.

You also give your environments team the opportunity to test their build process.

It is sometimes easy to forget that your performance testing tools need to run on a load injector machine, either physical or virtual, on premise or in the cloud.

Your load injector needs to be on the same network as your client base as you need to make sure you factor network latency into your performance test results.

This load injector needs to have enough CPU and Memory to support the performance testing which may, depending on your organisation, require you to simulate a lot of concurrent users.

If your load injector does not have enough CPU and/or Memory then this will affect your results and this may lead you to believe that your application is performing badly when in fact it is your load injector performing badly.

This can lead to time and effort being wasted search for performance bottlenecks in your application that do not exist.

This is where **OctoPerf** can help as they provide a service to allow you to test your application at scale on their dedicated hardware.

Their experienced team can also offer guidance on best practise and approach.

To understand the services they offer visit their product page.

https://octoperf.com/request-demo/

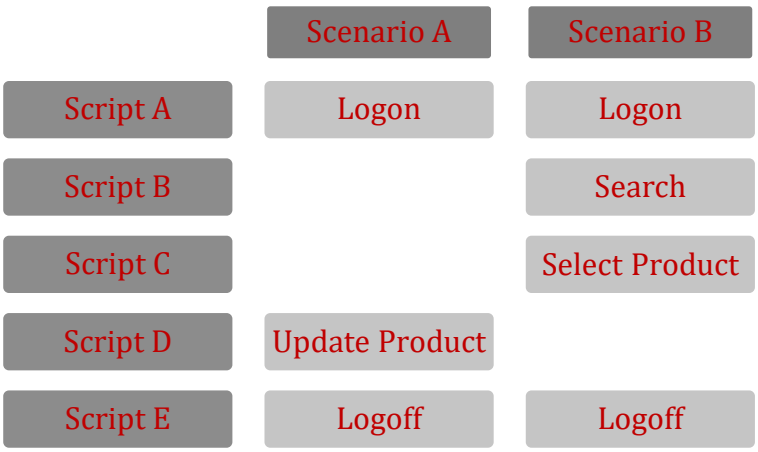
Reusability and Modularity

These principles have been mentioned in this paper but are worth further discussion as you can really add value in the way you create your performance tests and the purposes your tests fulfil.

The two concepts have common element but are different enough to be discussed separately.

Modularity is the concept of building complex business user journeys by using multiple scripts that each covers a small part of the journey.

By writing scripts that cover a small component in your application can benefit you in a number of ways:



- You can target specific parts of your application otherwise known as modularisation.
- You can chain these tests together to build complex user journeys.
- If a component changes you only have to maintain one test and not multiple tests.

Resusability is the concept of being able to use your tests for multiple purposes and to satisfy multiple requirements.

Examples of reusability are:

- Executing your tests with a number or users and levels of concurrency over a period of time to support a Peak Volume Load test.
- Executing the same test with varying users and levels of concurrency over the same period of time to support a Scalability test.
- Executing the same test with less users and concurrency over a long period of time to support a Soak test.

There is no need to have one test for each of these scenarios, where the distinction between them is the load volumes and duration of the test.

You use the same tests and drive them with alternative data parameters.

The concept of driving tests with data files is called abstraction and is too complicated for this beginners guide, a good guide can be found on the OctoPerf Blog pages.

https://octoperf.com/blog/2019/01/14/flexible-test-plans/

Benefits of reusability outside the execution of your performance testing include:

- Ability to create data in large quantities for the purposes of testing.
- Measure of system availability in the form of a sanity or smoke test.
- System reconciliation activities.

Really what we are encouraging is to provide other benefits with your performance testing assets and this is another really good reason to keep the performance testing capability in house and something an external consultancy would not do.

By thinking about reusability and modularity as part of your performance testing you are making sure that your tests will have wider programme benefits outside of their main purpose of performance test execution.

Document Results

We have already discussed documentation in terms of what we need to produce and the recommendation was not to produce anything except your requirements in the analysis stage but we made it clear that results will need to be produced.

You need to avoid the two greatest mistakes that many people make when producing results:

- That the programme managers and the business are interested in anything apart from assurances that it performs.
- Just producing page upon page of meaningless graphs around CPU, Memory, etc. just to arrive at the point above.

We need to remember that we are executing our tests on a regular basis and would therefore spend the majority of our time writing up our results.

The recommendation of this paper is to determine if your tests meet your requirements, if they have then mark the requirements as **Passed** or **Green** or whatever scale you use.

A test that passes one day may fail the next due to a code change so the constant execution cycles will track this.

If test fail to meet their requirement raise a defect or tech task, whatever is simplest and whatever works for your organisation.

If all your requirements are met at the end of the delivery cycles then you move to production.

Your requirements can then be constantly monitored as you execute on a regular basis where you may see regression or improvement but you are always aware of how your application performs.

It is important to understand that your requirements might be inaccurate when you get to execution as they were defined early in the programme and based on technology reasons or constraints may need to be reconsidered.

This is not a bad thing as regularly checking your requirements are still valid is good practise.

If a response time cannot be met under load, due to a technology constraint, as long as the stakeholders know you have tried to meet it and are happy with a change to the requirements then make the change.

Do not waste time trying to meet unachievable targets but equally do not just change your requirements just because the path to improvement seems difficult and time consuming.

Remember, any changes or addition or removal or requirements should follow the same approach as when they were originally defined and use the same audience for approval.

Conclusion

Performance testing should be simple, straightforward and quick we should be avoiding the out-dated approach that sees performance testing:

- over engineered,
- over documented,
- executed very late in the programme,
- with very little structure.

Make your performance testing agile even if the programme you are working within is not using these principles.

We are promoting an approach that avoids all unnecessary documentation and reporting as we see this as an outdated way of approaching performance testing.

Our message is that as your performance testing will be successful as long as you are testing the right functionality at the right volumes and concurrency against a representative environment measuring against good requirements.

By creating a simple straightforward approach that provides the opportunity to deliver and develop a performance testing capability over the period of the project rather than when timescales are incredibly tight is the objective of this document.

The reasons that performance testing is sometimes overlooked is because it is seen as a complex and time-consuming process that does not provide benefit.

It is common to run the performance testing too late with no time allowed for the process to provide benefit, which it invariably would, given the time and opportunity to do so.

You have the opportunity to change this philosophy if you consider some of the strategic points outlined in this white paper and that can only be a good thing.

You may still think that your organisation is too immature in terms of its delivery process to even attempt it but I urge you to try as it really will pay dividends.

You may be wholly outsourcing the build, quality assurance and delivery to a third party but that does not mean you should not define your performance testing requirements and give yourself something to hold your supplier accountable to.