# Good pratices in performance testing

José Esteves, Student Member, TVVS, Software Engineering, FEUP,
Ricardo Malafaya, Student Member, TVVS, Software Engineering, FEUP.

***Abstract*—The Performance testing is a complex activity that cannot effectively be shaped into a "one-size-fits-all" or even a "one-size-fits-most" approach. Although the complexity of the process and the diversity of software projects there are a set of core activities that are essential. In this paper, we focus on identifying each core activity by demonstrating its meaning and applicability with practical tips and guides. Furthermore, the paper will present good practices for managing performance tests in different developing environments like waterfall, agile and safety-critical systems. This paper hence provides a comprehensive guide of good practices of the core process of performance testing and the different approaches that it can take.**

***Keywords—performance testing; good pratices; testing activities; manage testing***

## I. Introduction

The performance testing has as its goal to assess the throughput, responsiveness, reliability, scalability of system under a workload.
The focus of performance testing is:

- Determine if a system is ready by predicting performance characteristics.
- Identify if the infrastructure is adequate by determining the system's stability and capacity.
- Compare software performance between different builds and the desire performance characteristics.
- Analyze the capacity of the systems under different workloads.
- Identify bottlenecks of the applications.

A possible performance approach is divide in seven activities.

- *Identify the Test Environment* - Identify the physical test environment and the production environment as well as the tools and resources available to the test team. The physical environment includes hardware, software, and network configurations. Having a thorough understanding of the entire test environment at the outset enables more efficient test design and planning.
- *Identify Performance Acceptance Criteria* - Identify the response time, throughput, and resource utilization goals and constraints. In general, response time is a user concern, throughput is a business concern, and resource utilization is a system concern.
- *Plan and Design Test* - Identify key scenarios, determine variability among representative users and how to simulate that variability, define test data, and establish metrics to be collected.
- *Configure the Test Environment* - Prepare the test environment, tools, and resources necessary to execute each strategy as features and components become available for test.
- *Implement the Test Design* - Develop the performance tests in accordance with the test design.
- *Execute the Test* - Run and monitor the tests. Validate the tests, test data, and results collection. Execute validated tests for analysis while monitoring the test and the test environment.
- *Analyze Results, Report, and Retest* - Consolidate and share results data. Analyze the data both individually and as a cross-functional team. Reprioritize the remaining tests and re-execute them as needed.

Performance tests are usually described as belonging to one of the following three categories:

- *Performance testing* - This type of testing determines or validates the speed, scalability, and/or stability characteristics of the system or application under test. Performance is concerned with achieving response times, throughput, and resource utilization levels that meet the performance objectives for the project or product.
- *Load testing* - This subcategory of performance testing is focused on determining or validating performance characteristics of the system or application under test when subjected to workloads and load volumes anticipated during production operations.
- *Stress testing* - This subcategory of performance testing is focused on determining or validating performance characteristics of the system or application under test when subjected to conditions beyond those anticipated during production operations. Stress tests may also include tests focused on determining or validating performance characteristics of the system or application under test when subjected to other stressful conditions, such as limited memory, insufficient disk space, or server failure. These tests are designed to determine under what conditions an application will fail, how it will fail, and what indicators can be monitored to warn of an impending failure.

When end-to-end performance testing reveals system or application characteristics that are deemed unacceptable, test teams change their focus from performance testing to performance tuning, to discover what is necessary to make the application perform acceptably. Although tuning is not the direct responsibility of most performance testers, the tuning process is most effective when it is a cooperative effort between all of those concerned with the application or system under test. The performance testing results can be acceptable but still want to focus on tuning to reduce the amount of resources being used.

**Table 1 - Matrix of Benefits by Key Performance Test Types**

| *Term* | *Benefits* | *Challenges and Areas Not Addressed* |
|---|---|---|
| Performance test | Determines the speed, scalability and stability characteristics of an application. Focuses on determining if the user of the system will be satisfied with the performance characteristics of the application. Supports tuning, capacity planning, and optimization efforts. | May not detect some functional defects that only appear under load. If not carefully designed and validated, may only be indicative of performance characteristics in a very small number of production scenarios. |
| Load test | Determines the throughput required to support the anticipated peak production load and determines the adequacy of a hardware environment. Helps to determine how much load the hardware can handle before resource utilization limits are exceeded. | Is not designed to primarily focus on speed of response. Results should only be used for comparison with other related load tests. |
| Stress test | Determines if data can be corrupted by overstressing the system. Provides an estimate of how far beyond the target load an application can go before causing failures and errors in addition to slowness. Allows to establish application-monitoring triggers to warn of impending failures. | It is often difficult to know how much stress is worth applying. It is possible to cause application and/or network failures that may result in significant disruption if not isolated to the test environment. |

Creating a baseline is the process of running a set of tests to capture performance metric data for the purpose of evaluating the effectiveness of subsequent performance-improving changes to the system or application. A critical aspect of a baseline is that all characteristics and configuration options except those specifically being varied for comparison must remain invariant. The baseline is used to compare against the system performance or against an industry standard. This process is called benchmarking.

It is important to understand the different performance test types in order to reduce risks, minimize cost, and know when to apply the appropriate test over the course of a given performance-testing project. In Table 1 is presented the benefits by key performance test types.

Performance testing is indispensable for managing certain significant business risks. For example, if a Web site cannot handle the volume of traffic it receives, the customers will shop somewhere else. Beyond identifying the obvious risks, performance testing can be a useful way of detecting many other potential problems. While performance testing does not replace other types of testing, it can reveal information relevant to usability, functionality, security, and corporate image that is difficult to obtain in other ways.

## II. PERFORMANCE TESTING APPROACHES

Performance testing is a complex activity that cannot effectively be shaped into a "one-size-fits-all" or even a "one-size-fits-most" approach. Projects, environments, business drivers, acceptance criteria, technologies, timelines, legal implications, and available skills and tools simply make any notion of a common, universal approach unrealistic. Is always important to understand that performance testing has

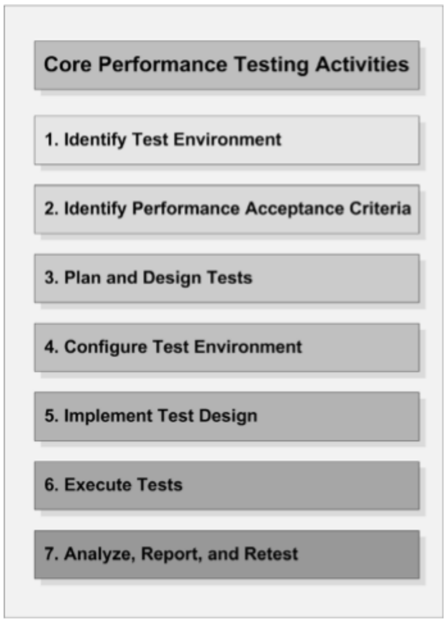
Core Performance Testing Activities
1. Identify Test Environment
2. Identify Performance Acceptance Criteria
3. Plan and Design Tests
4. Configure Test Environment
5. Implement Test Design
6. Execute Tests
7. Analyze, Report, and Retest

**Figure 1 - Core performance testing activities**

to be applied in a way that best fits the project context. The core activities can be seen in the same way. They are presented as a sequence but some can be done in parallel. Depending on the size and complexity of the project, an iterative test cycle of some of these steps can be implemented. The core activities can be seen in Figure 1.

### *A. Identify the Test Environment*

*1) Input*

*a)* Logical and physical production architecture.

*b)* Logical and physical test architecture.

*c)* Available tools.

*2) Output*

*a)* Comparison of test and production environments.

*b)* Environment-related concerns.

*c)* Determination of whether additional tools are required.

The environment in which the performance tests will be executed, along with the tools and associated hardware necessary to execute the performance tests, constitute the test environment. Under ideal conditions, if the goal is to determine the performance characteristics of the application in production, the test environment is an exact replica of the production environment but with the addition of load-generation and resource-monitoring tools. The key factor in identifying the test environment is to completely understand the similarities and differences between the test and production environments. Some critical factors to consider are:

- Hardware - configurations and machine characteristics.
- Network - configuration, end-user network location and architecture
- Tools - load generation tool limitations and impact of monitoring tools.
- Software - already running on machine, storage capacity, logging levels.
- External factors - additional traffic on network, updates and backups in process and interactions with other systems.

### *B. Identify Performance Acceptance Criteria*

*1) Input*

*a)* Client expectations.

*b)* Risks to be mitigated.

*c)* Business requirements.

*2) Output*

*a)* Performance-testing success criteria.

*b)* Performance goals and requirements.

*c)* Key areas of investigation.

*d)* Key performance  and business indicators.

In order to access the capabilities of the system, it generally makes sense to start identifying, or at least estimating, the desired performance characteristics of the application early in the development life cycle. This can be accomplished most simply by noting the performance characteristics that users and stakeholders equate with good performance. The classes of characteristics that are frequently used are:

- Response time - For example, the product catalog must be displayed in less than three seconds.
- Throughput - For example, the system must support 25 book orders per second.
- Resource utilization - For example, processor utilization is not more than 75 percent. Other important resources that need to be considered for setting objectives are memory, disk input/output (I/O), and network I/O.

### *C. Plan and Design Test*

*1) Input*

*a)* Available application features and/or components.

*b)* Application usage scenarios.

*c)* Unit tests.

*d)* Performance acceptance criteria.

*2) Output*

*a)* Conceptual strategy.

*b)* Test execution prerequisites.

*c)* Tools and resources required.

*d)* Application usage models to be simulated.

*e)* Test data required to implement tests.

Planning and designing performance tests involves identifying key usage scenarios, determining appropriate variability across users, identifying and generating test data, and specifying the metrics to be collected. Key usage scenarios for the application typically surface during the process of identifying the desired performance characteristics of the application. When identified, captured, and reported correctly, metrics provide information about how application's performance compares to desired performance characteristics. In addition, metrics can help to identify problem areas and bottlenecks within the application. Consider the following key points when planning and designing tests:

- Realistic test designs are real-world simulations that are sensitive to dependencies outside the control of the system, such as humans, network activity, and other systems interacting with the application.
- Realistic test designs can be more costly and time-consuming to implement, but they provide far more accuracy for the business and stakeholders.
- Involve the developers and administrators in the process of determining which metrics are likely to add value and which method best integrates the capturing of those metrics into the test.

- Beware of allowing tools to influence the test design. Better tests almost always result from designing tests on the assumption that they can be executed and then adapting the test or the tool when that assumption is proven false, rather than by not designing particular tests based on the assumption that the access to a tool to execute the test is missing.

### D. *Configure the Test Environment*

#### 1) *Input*

*a)* Conceptual strategy.

*b)* Available tools.

*c)* Designed tests.

#### 2) *Output*

*a)* Configured load-generation and resource-monitoring tools.

*b)* Environment ready for performance testing.

Preparing the test environment, tools, and resources for test design implementation and test execution prior to features and components becoming available for test can significantly increase the amount of testing that can be accomplished during the time those features and components are available. Additionally, plan to periodically reconfigure, update, add to, or otherwise enhance the load-generation environment and associated tools throughout the project. Consider the following key points when configuring the test environment:

- Determine how much load can be generated before the load generators reach a bottleneck. Typically, load generators encounter bottlenecks first in memory and then in the processor.
- Check for any inconsistent or error in the configuration of test environment. Doing so can save significant time and prevent from having to dispose of the data entirely and repeat the tests.
- Validate the accuracy of load test execution against hardware components such as switches and network cards and related to server clusters in load-balanced configuration.
- Monitor resource utilization (CPU, network, memory, disk and transactions per time) across servers in the load-balanced configuration during a load test to validate that the load is distributed.

### E. *Implement the Test Design*

#### 1) *Input*

*a)* Conceptual strategy.

*b)* Available tools.

*c)* Designed tests.

#### 2) *Output*

*a)* Configured load-generation and resource-monitoring tools.

*b)* Environment ready for performance testing.

Regardless of the tool, creating a performance test typically involves scripting a single usage scenario and then enhancing that scenario and combining it with other scenarios to ultimately represent a complete workload model.
Consider the following key points when implementing the test design:

- Ensure that test data feeds are implemented correctly.
- Ensure that application data feeds are implemented correctly in the database and other application components.
- Ensure that validation of transactions is implemented correctly.
- Ensure hidden fields or other special data are handled correctly.
- Add pertinent indicators to facilitate articulating business performance.
- It is generally worth taking the time to make the script match the designed test, rather than changing the designed test to save scripting time.
- Significant value can be gained from evaluating the output data collected from executed tests against expectations in order to test or validate script development.

### F. *Execute Tests*

#### 1) *Input*

*a)* Task execution plan.

*b)* Available tools/environment.

*c)* Available application features and/or components.

*d)* Validated, executable tests.

#### 2) *Output*

*a)* Test execution results.

Executing tests is what most people envision when they think about performance testing. It makes sense that the process, flow, and technical details of test execution are extremely dependent on the tools, environment, and project context. Test execution can be viewed as a combination of the following sub-tasks:

- Coordinate test execution and monitoring with the team.
- Validate tests, configurations, and the state of the environments and data.
- Begin test execution.

- While the test is running, monitor and validate scripts, systems, and data.
- Upon test completion, review the results for obvious indications that the test was flawed.
- Archive the tests, test data, results, and other information necessary to repeat the test later if needed.
- Log start and end times, the name of the result data, and so on. This will allow to identify data sequentially after the test is done.

Consider the following key points when executing the test:

- Validate test executions for data updates, such as orders in the database that have been completed.
- Validate if the load-test script is using the correct data values, such as product and order identifiers, in order to realistically simulate the business scenario.
- Whenever possible, limit test execution cycles to one to two days each. Review and reprioritize after each cycle.
- Do not process data, write reports, or draw diagrams on the load-generating machine while generating a load, because this can skew the results of the test. Ensure that any other process running on the machine not relevant to the test is terminated.
- While load is being generated, access the system manually from a machine outside of the load-generation environment during test execution so that observations and the results data can be compared at a later time.
- Simulate ramp-up and cool-down periods appropriately.

### G. *Analyze Results, Report, and Retest*

1) *Input*
   *a)* Task execution results.
   *b)* Performance acceptance criteria.
   *c)* Risks, concerns, and issues.
2) *Output*
   *a)* Results analysis.
   *b)* Recommendations.
   *c)* Reports.

Managers and stakeholders need more than just the results from various tests — they need conclusions, as well as consolidated data that supports those conclusions. Technical team members also need more than just results — they need analysis, comparisons, and details behind how the results were obtained. Team members of all types get value from performance results being shared more frequently.

Most reports fall into one of the following two categories:

- Technical Reports - Description of the test, including workload model and test environment. Easily digestible data with minimal pre-processing with providing access to the complete data set and test conditions. Provides statements of observations, concerns, questions, and requests for collaboration.
- Stakeholder Reports - Description if the results with visual representations of the most relevant data and the criteria to which the results relate. Provides representations of the workload model and test environment and summaries of observations, concerns, and recommendations. Permits an access to associated technical reports, complete data sets, and test conditions.

Consider the following important points when analyzing the data returned by the performance test:

- Analyze the data both individually and as part of a collaborative, cross-functional technical team. Report visually.
- Analyze the captured data and compare the results against the metric's acceptable or expected level to determine whether the performance of the application being tested shows a trend toward or away from the performance objectives. Use the right statistics.
- If the test fails, a diagnosis and tuning activity are generally warranted.
- Performance-testing results will often enable the team to analyze components at a deep level and correlate the information back to the real world with proper test design and usage analysis.
- Performance test results should enable informed architecture and business decisions.
- Frequently, the analysis will reveal that, in order to completely understand the results of a particular test, additional metrics will need to be captured during subsequent test execution cycles.

## III. Managing the Performance Test Cycle

Performance testing is a critical aspect of many software projects because it tests the architectural aspects of the customer experience and provides an indication of overall software quality. Because it is frequently expensive to set up and integrate performance testing, project teams often wait until the end of the project development/test life cycle to do so. The potential side effect to this approach is that when major issues are found near the end of the development life cycle, it becomes much more expensive to resolve them. The key to working within an iteration-based work cycle is team coordination. For this reason, the performance tester must be able to adapt what he or she measures and analyzes per iteration cycle as circumstances change.

## *A. Managing the Performance Test Cycle in Waterfall*

This approach can be represented by using the following nine activities (Figure 2).

- *Activity 1* - Understand the Project Vision and Context. The outcome of this activity is a shared understanding of the project vision and context.
- *Activity 2* - Identify Reasons for Testing Performance. Explicitly identify the reasons for performance testing.
- *Activity 3* - Identify the Value Performance Testing Adds to the Project  Translate the project- and business-level objectives into specific, identifiable, and manageable performance-testing activities.
- *Activity 4* - Configure the Test Environment. Set up the load-generation tools and the system under test, collectively known as the performance test environment.
- *Activity 5* - Identify and Coordinate Tasks. Prioritize and coordinate support, resources, and schedules to make the tasks efficient and successful.
- *Activity 6* - Execute Task(s). Execute the activities for the current iteration.

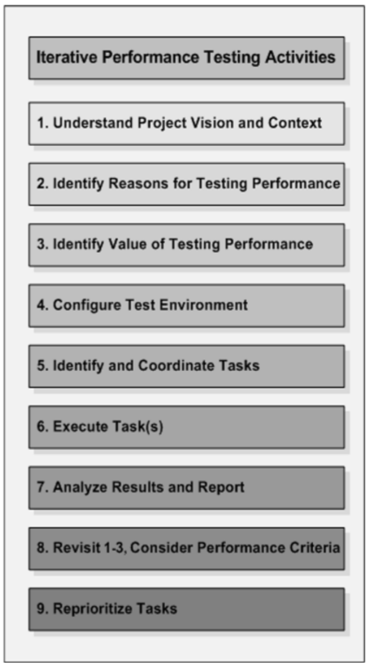
Iterative Performance Testing Activities
1. Understand Project Vision and Context
2. Identify Reasons for Testing Performance
3. Identify Value of Testing Performance
4. Configure Test Environment
5. Identify and Coordinate Tasks
6. Execute Task(s)
7. Analyze Results and Report
8. Revisit 1-3, Consider Performance Criteria
9. Reprioritize Tasks

**Figure 2 - Iterative Performance Testing Activities in Waterfall**

- *Activity 7* - Analyze Results and Report. Analyze and share results with the team.
- *Activity 8* - Revisit Activities 1-3 and Consider Performance Acceptance Criteria. Between iterations, ensure that the foundational information has not changed. Integrate new information such as customer feedback and update the strategy as necessary.
- *Activity 9* - Reprioritize Tasks. Based on the test results, new information, and the availability of features and components, reprioritize, add to, or delete tasks from the strategy, and then return to activity 5.

The Figure 3 displays how the seven core activities described map to these nine activities.

## *B. Managing the Performance Test Cycle in Agile*

This approach can be represented by using the following nine activities (Figure 4).

- *Activity 1* - Understand the Project Vision and Context.  The project vision and context are the foundation for determining what performance-testing activities are necessary and valuable.
- *Activity 2* - Identify Reasons for Testing Performance.  These are not always clear from the vision and context. Explicitly identifying the reasons for performance testing is critical to being able to determine what performance testing activities will add the most value to the project.
- *Activity 3* - Identify the Value Performance Testing. Adds to the Project.  With the information gained from steps 1 and 2, clarify the value added through performance testing and convert that value into a conceptual performance-testing strategy.

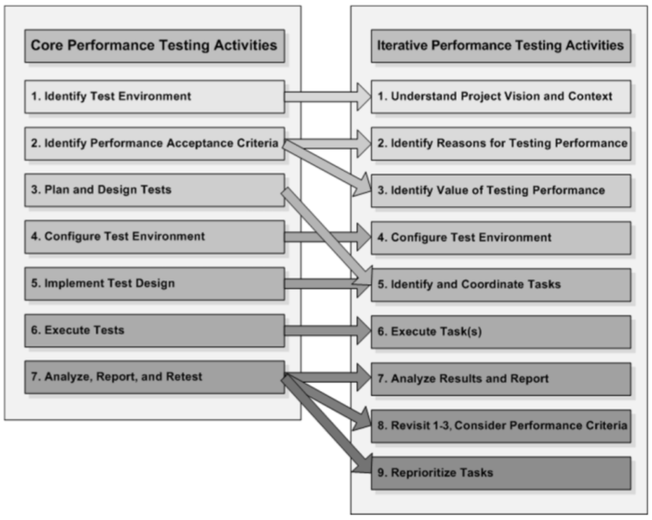
Core Performance Testing Activities
1. Identify Test Environment
2. Identify Performance Acceptance Criteria
3. Plan and Design Tests
4. Configure Test Environment
5. Implement Test Design
6. Execute Tests
7. Analyze, Report, and Retest
Iterative Performance Testing Activities
1. Understand Project Vision and Context
2. Identify Reasons for Testing Performance
3. Identify Value of Testing Performance
4. Configure Test Environment
5. Identify and Coordinate Tasks
6. Execute Task(s)
7. Analyze Results and Report
8. Revisit 1-3, Consider Performance Criteria
9. Reprioritize Tasks

**Figure 3 - Relationship to Core Performance Testing Activities in Waterfall**

- *Activity 4* - Configure the Test Environment.  With a conceptual strategy in place, prepare the necessary tools and resources to execute the strategy as features and components become available for test.

- *Activity 5* - Identify and Coordinate Immediately Valuable Tactical Tasks. Performance testing tasks do not happen in isolation. For this reason, the performance specialist needs to work with the team to prioritize and coordinate support, resources, and schedules in order to make the tasks efficient and successful.

- *Activity 6* - Execute Task(s). Conduct tasks in one- to two-day segments. See them through to completion, but be willing to take important detours along the way if an opportunity to add additional value presents itself.

- *Activity 7* - Analyze Results and Report. To keep up with an iterative process, results need to be analyzed and shared quickly. If the analysis is inconclusive, retest at the earliest possible opportunity. This gives the team the most time to react to performance issues.

- *Activity 8* - Revisit Activities 1-3 and Consider Performance Acceptance Criteria. Between iterations, ensure that the foundational information has not changed. Integrate new information such as customer feedback and update the strategy as necessary.

- *Activity 9* - Reprioritize Tasks. Based on the test results, new information, and the availability of features and components, reprioritize, add to, or delete tasks from the strategy, then return to activity.

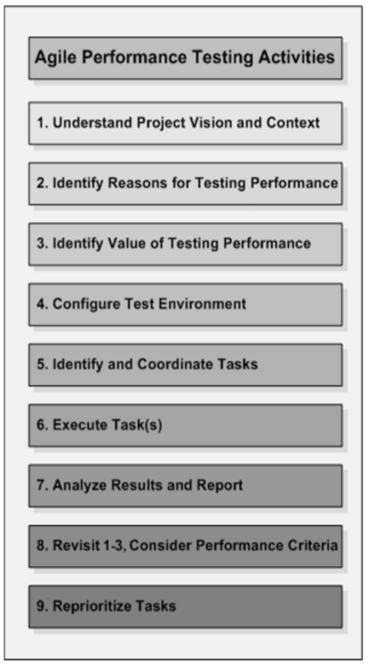
Agile Performance Testing Activities
1. Understand Project Vision and Context
2. Identify Reasons for Testing Performance
3. Identify Value of Testing Performance
4. Configure Test Environment
5. Identify and Coordinate Tasks
6. Execute Task(s)
7. Analyze Results and Report
8. Revisit 1-3, Consider Performance Criteria
9. Reprioritize Tasks

**Figure 4 - Iterative Performance Testing Activities in Agile**

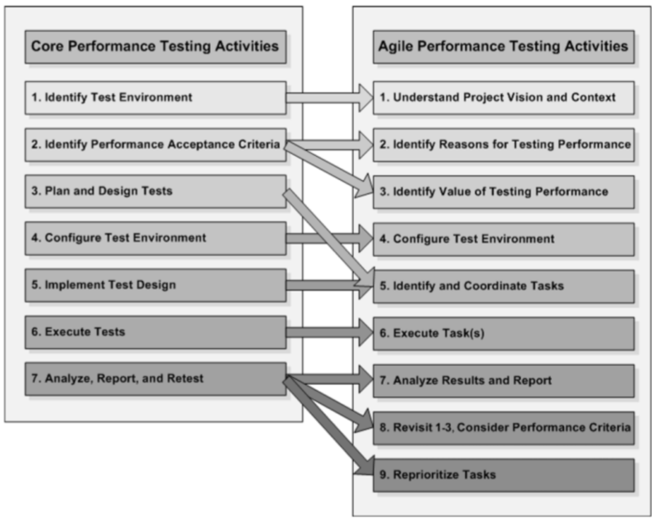
Core Performance Testing Activities
1. Identify Test Environment
2. Identify Performance Acceptance Criteria
3. Plan and Design Tests
4. Configure Test Environment
5. Implement Test Design
6. Execute Tests
7. Analyze, Report, and Retest
Agile Performance Testing Activities
1. Understand Project Vision and Context
2. Identify Reasons for Testing Performance
3. Identify Value of Testing Performance
4. Configure Test Environment
5. Identify and Coordinate Tasks
6. Execute Task(s)
7. Analyze Results and Report
8. Revisit 1-3, Consider Performance Criteria
9. Reprioritize Tasks

**Figure 5 - Relationship to Core Performance Testing Activities in Agile**

The Figure 5 displays how the seven core activities described map to these nine activities.

### C. *Managing the Performance Test Cycle in safety-critical software*

This approach can be represented by using the following nine activities (Figure 6).

- *Activity 1* - Understand the Process and Compliance Criteria. This activity involves building an understanding of the process and the compliance requirements.

- *Activity 2* - Understand the System and the Project Plan. Establish a fairly detailed understanding of the system to test and the project specifics for the development of that system.

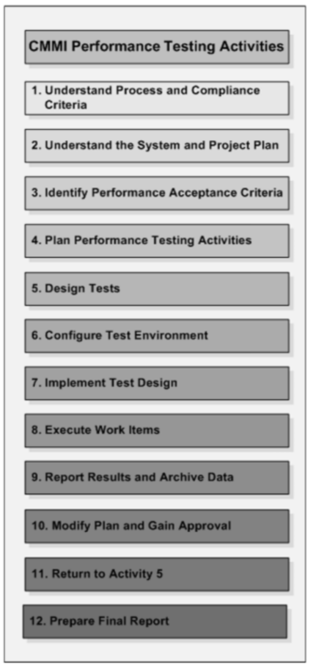
CMMI Performance Testing Activities
1. Understand Process and Compliance Criteria
2. Understand the System and Project Plan
3. Identify Performance Acceptance Criteria
4. Plan Performance Testing Activities
5. Design Tests
6. Configure Test Environment
7. Implement Test Design
8. Execute Work Items
9. Report Results and Archive Data
10. Modify Plan and Gain Approval
11. Return to Activity 5
12. Prepare Final Report

**Figure 6 - Iterative Performance Testing Activities in safety-critical software**

- *Activity 3* - Identify Performance Acceptance Criteria. This activity includes identify the performance goals and requirements. This also includes identifying the performance testing objectives.
- *Activity 4* - Plan Performance-Testing Activities. This activity includes mapping work items to the project plan, determining durations, prioritizing the work, and adding detail to the plan.
- *Activity 5* - Design Tests. This activity involves identifying key usage scenarios, determining appropriate user variances, identifying and generating test data, and specifying the metrics to be collected.
- *Activity 6* - Configure the Test Environment. This activity involves setting up the actual test environment.
- *Activity 7* - Implement the Test Design. This activity involves creating the tests.
- *Activity 8* - Execute Work Items. This activity involves executing the performance test work items.
- *Activity 9* - Report Results and Archive Data. This activity involves consolidating results and sharing data among the team.
- *Activity 10* - Modify the Plan and Gain Approval for Modifications. This activity involves reviewing and adjusting the plan as needed.
- *Activity 11* - Return to Activity 5. This activity involves continuous testing through the next delivery, iteration, and checkpoint release.
- *Activity 12* - Prepare the Final Report. This activity involves the creation, submission, and acceptance of the final report.

The Figure 7 displays how the seven core activities described map to these nine activities.

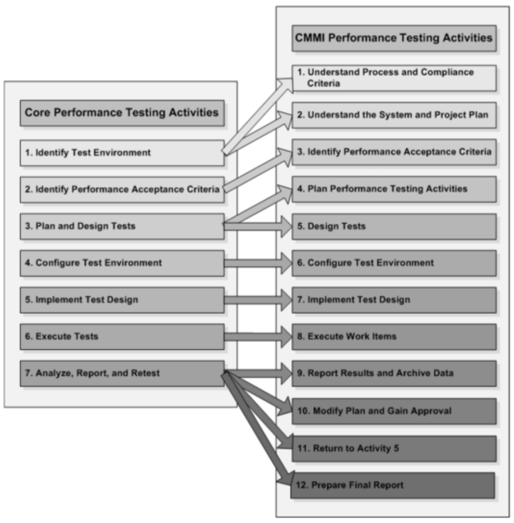
Core Performance Testing Activities
1. Identify Test Environment
2. Identify Performance Acceptance Criteria
3. Plan and Design Tests
4. Configure Test Environment
5. Implement Test Design
6. Execute Tests
7. Analyze, Report, and Retest
CMMI Performance Testing Activities
1. Understand Process and Compliance Criteria
2. Understand the System and Project Plan
3. Identify Performance Acceptance Criteria
4. Plan Performance Testing Activities
5. Design Tests
6. Configure Test Environment
7. Implement Test Design
8. Execute Work Items
9. Report Results and Archive Data
10. Modify Plan and Gain Approval
11. Return to Activity 5
12. Prepare Final Report

**Figure 7 - Relationship to Core Performance Testing Activities in safety-critical software**

## IV. TESTING TOOLS

The aid of testing tools is essential in plugging the gap between short deadlines and the time spent in manual testing, in enhancing the reliability of the software as well as increasing the efficiency of the those who produce it. All this is magnified by the always-changing nature of the software development environment, which manifests itself on the competitiveness within that industry. Testing tools are a bridge that connects the need of fast production and quality assurance. Here are some examples of performance testing tools.

### *A. WebLOAD*

It is a web and mobile load testing and analysis tool that combines performance, scalability and integrity as a single process for the verification of those same platforms applications. This tool is focused on giving clear information on the functionality and the actual capacity of the application.

### *B. LoadUI NG Pro*

LoadUI NG Pro is an open-source performance testing tool that provides a graphical interface and it is most effective when used jointly with soapUI. This tool aims for a complemented analytical approach of testing and promotes the use of its reports and advanced analysis features.

### *C. Apica LoadTest*

Apica LoadTest offers two types of solutions, one self-service and another full-service. It allows a test on-demand option or an automated testing one throughout the development lifecycles.

### *D. Apache JMeter*

This tool is a Java platform application. Apache JMeter can be used to test performance both on static and dynamic resources and has the capacity to be loaded into a server or network, so as to check on its performance and analyze its working under different conditions.

### *E. HP LoadRunner*

HP LoadRunner is useful in understanding and determining the performance and outcome if the system. As this tool can create and support thousands of users, it enables to gather the required information in respect to the performance and also based on the infrastructure.

### *F. Rational Performance Tester*

It is an automated performance testing tool which can be used for a web application or a server based application where there is a process of input and output is involved. This tool is well adapted to the process of building an effective free cloud computing service.

### *G. NeoLoad*

NeoLoad is directed at testing the performance of websites. It does so by increasing the traffic of a said website.

### *H. LoadComplete*

LoadComplete allows you to generate and execute realistic load tests for web applications. It helps you check your web server's performance under a massive load and estimate its scalability by simulating user actions.

### *I. WAPT*

Another web application performance testing tool, WAPT focuses on testing any web service or web interface of the application so it can test the system under a diverse set of environments and load conditions.

### *J. Loadster*

Loadster is an HTTP load testing tool and it is best used to detect bottlenecks in the application.

### *K. Load Impact*

This tool is targets the load testing of cloud-based services and helps to develop the optimization of a web application. It offers a page analyzer to support the report and study of the system parameters.

### *L. Appvance UTP*

Appvance UTP is a unified software test automation platform. It unifies tests with a write-once methodology which means a functional test can be re-used for performance, load, compatibility, app-penetration and synthetic APM.

### *M. Testing Anywhere*

Yet another web application testing tool, it can be used to test the performance of a website and identify, as well as rectify, bottlenecks within the system.

### *N. OpenSTA*

OpenSTA is a complex GUI based performance testing tool capable of performing heavy load tests and analyses for the scripted HTTP and HTTPS.

### *O. QEngine*

QEngine is an easy-to-use automate testing tool used in performance, load testing and has other options such as functionality testing, compatibility testing, and stress testing. Its most appealing feature is its ability to perform remote testing of web services from any geographical location.

## V. CONCLUSION

In this paper, we analyzed the solutions currently available for the design, planning and execution of performance tests. The discussed activities and approaches are not standardized but this paper offers a base set of core activities as a good practice to follow and how to prevent fall in some common and familiar mistakes. The range of projects and development process does not permit to create a universal approach. Instead, in this paper, we demonstrate how to manage the core activities of performance testing into different productions and projects types. This paper is a broader understanding of what all the factors might be so that reader have an excellent menu of considerations to choose from when creating and execute a testing plan.

## REFERENCES

[1] J. Meier, C. Farre *et al.*, "Performance Testing Guidance for Web Applications", Microsoft, 2007.

[2] S. Sharmila and E. Ramadevi, "Analysis of Performance Testing on Web Applications", International Journal of Advanced Research in Computer and Communication Engineering, Vol. 3, Issue 3, 2014, p. 5258-5260.

[3] H. Sarojadevi, "Performance Testing: Methodologies and Tools", Journal of Information Engineering and Applications, Vol 1, No. 5, 2011, p. 5-12.

[4] http://www.softwaretestinghelp.com/performance-testing-tools.