Playbook Methods Repository

# **Technical Performance Review**

Evaluate the technical performance metrics of a product across targeted devices and use cases. Sample metrics include responsiveness, memory usage, battery usage, load time, and peak throughput.

### Remote Agility: **•** High

### Linked Tactic(s): Product Performance Analysis

## **Why we do it:**

The features and functionality of a software system are not the only considerations. The performance of a software program, such as its reaction time, dependability, resource utilization, and scalability, is critical. Performance Review aims not to uncover problems but to reduce performance bottlenecks.

Technical performance review, also called Performance Testing, is done to offer stakeholders with information about their application regarding speed, stability, and scalability. More significantly, Performance Testing exposes what has to be addressed before the product is marketed. Without Performance Testing, the software is likely to suffer from difficulties such as: running sluggishly when numerous people use it concurrently, inconsistencies across various operating systems, and poor usability.

Performance testing will verify whether their program satisfies speed, scalability, and stability criteria under predicted workloads. Applications released to the market with low-performance metrics owing to nonexistent or inadequate performance testing are likely to earn a negative reputation and fail to fulfill planned sales targets.

## When to apply it:

* Xx

## Best Practices & Considerations:

* Test Early and Often: Take an agile approach that uses iterative testing throughout the entire development life cycle. Specifically, provide the ability to run performance “unit” testing as part of the development process – and then repeat the same tests on a larger scale in later stages of application readiness.
* Consider Users, Not Just Servers : Performance tests often focus solely on the performance of servers and clusters running software. Don’t forget that people use software, and performance tests also should measure the human element.
* Understand Performance Test Definitions: It’s crucial to have a common definition for the types of performance tests that should be executed against your applications, such as:
  + Single User Tests: Testing with one active user yields the best possible performance, and response times can be used for baseline measurements.
  + Load Tests: Understand the behavior of the system under average load, including the expected number of concurrent users performing a specific number of transactions within an average hour.
  + Peak Load Tests: Understand system behavior under the heaviest anticipated usage for concurrent number of users and transaction rates.
  + Endurance (Soak) Tests: Determine the longevity of components, and whether the system can sustain average to peak load over a predefined duration.
  + Stress Tests: Understand the upper limits of capacity within the system by purposely pushing it to its breaking point.
  + High Availability Tests: Validate how the system behaves during a failure condition while under load. There are many operational use cases that should be included, such as seamless failover of network equipment or rolling server restarts.
* Build a Complete Performance Model: Measuring your application’s performance includes understanding your system’s capacity. This includes planning what the steady state will be in terms of concurrent users, simultaneous requests, average user sessions and server utilization during peak periods of the day
* Measure Averages, but Include Outliers: When testing performance, you need to know average response time, but this measurement can be misleading by itself. Be sure to include other metrics, such as 90th percentile or standard deviation, to get a better view of system performance. KPIs can be measured by looking at average and standard deviations.
* Consistently Report and Analyze the Results : Reports communicate the results of your application’s behavior to everyone in your organization, especially project owners and developers. Analyzing and reporting results consistently also helps to determine future updates and fixes.
* Triage Performance Issues: Providing the results of performance tests is fine, but those results, especially when they demonstrate failure, are not enough. The next step should be to triage the code/application and system performance, and involve all parties: developers, testers and operations personnel involved.

## Responsible roles:

* xx

## Tools:

### Online tools/platforms/services

* + LoadNinja, Apache JMeter, WebLOAD, LoadUI Pro, LoadView, LoadRunner, Silk Performer, AppLoader

### Websites

* + xx

### Databases

* + xx
* Other
  + xx

## 

## Thoughtworks Examples - Linked

### Client working docs, airtable, miro/mural boards

* + xx

### Client polished presentations/deliverables

* + xx

### Internal assets - clinic materials / guild docs

* + xx

## 

## Learn more: How we do this?

### Templates (docs, decks, sheets, miro, etc.)

* + xx

### How-To Resources (external or internal)

* + <https://www.guru99.com/performance-testing.html>
  + <https://devops.com/best-practices-for-application-performance-testing/>
  + <https://www.edureka.co/blog/performance-testing-tools/>

### Outside References (articles, books, etc.)

* + xx

### Sub-set Activities

* + xx

## 

## 