Risk-based Exploratory Performance Testing

Author: Giridhar Mangena
Deloitte Consulting India Private Limited
Block A, Deloitte Drive, Road No. 2, Hi-tech City Layout,
Hyderabad - 81, India

Author: Kalyan Kumar Manikonda
Deloitte Consulting India Private Limited
Block A, Deloitte Drive, Road No. 2, Hi-tech City Layout,
Hyderabad - 81, India

Author: Jitin Kulshrestha
Deloitte Consulting India Private Limited
Block A, Deloitte Drive, Road No. 2, Hi-tech City Layout,
Hyderabad - 81, India
Abstract:
Today's application development landscape has been consistently transforming to new heights to make everything complicated. Customers are always on the lookout to deliver their products and services to market with great quality and within short span of time. The end users in turn expect a greater response time from these products and services to perform their activities, which is influencing the customers to fine tune the performance of their products and applications. With the increased focus on performance, it is challenging to assess the right non-functional requirements and right Service Level Agreements (SLAs) to optimize the systems. Even though if Non-Functional Requirements (NFRs) and SLAs are defined, are they sufficient to say that the other components of the system will not break down or their performance won't get worse?

Exploratory testing is a powerful approach coined to improve testing outcomes without scripted testing. Most of us practice some form of exploratory testing day-to-day on the projects, however it is currently being limited to functional testing. This paper describes a practical approach to carry out exploratory performance testing in a risk based approach to improve the application response time, and reduce cost and resource utilization to a greater extent.

Key Takeaways:
- How performance testing can be done in an exploratory way?
- How is exploratory performance different from regular performance testing practice?

Addressed Audience:
Project managers, Performance testing practitioners and engineers, Test managers, QA managers, Development community, Infrastructure teams, as well as other professionals interested in building and delivering better software.
Introduction

Exploratory testing is sometimes confused with “Adhoc” testing. Adhoc testing normally refers to a process of improvised, impromptu bug searching. By definition, anyone can do adhoc testing, but the term “exploratory testing”—coined by Cem Kaner, in testing Computer Software refers to a sophisticated, thoughtful approach to adhoc testing. Till today, exploratory testing has been limited to functional testing, but is it really limited to that? Give a thought to this word “Exploratory” and imagine it in the world of performance testing. One of the biggest challenges in the world of performance testing is having clear “NFRs and SLAs”. We often see that the customer itself is not clear with his requirements pertaining to application performance. There may be many reasons for this starting from lack of historical data for the functionality involved to lack of required knowledge of the person representing the customer. It becomes much more challenging to perform performance testing when you do not have clear NFRs and SLAs.

Till now, in the traditional waterfall implementations, performance testing takes place at the end of the lifecycle and typically includes these steps:

- Prepare the system.
- Develop requested scripts.
- Run scripts in the requested combinations.
- Compare results with the requirements provided.
- Allow some percentage of errors according to the requirements.
- Publish the recommendation and report the results

Challenges

“What are the current challenges in Traditional and Agile performance testing models? What is exploratory performance testing and why is it Risk-based? Why is it needed and what is the exit criteria to say “performance testing” is completed and the system is performing within the limits?”

This approach to the performance testing appears to be a very well matured process. But there are many potential and serious problems as below:

- The waterfall approach assumes that the entire system—or at least all the functional components involved—is ready for the performance test. This means the performance testing can’t be done until very late in the development cycle, at this stage even small fixes would be expensive. Also it’s not feasible to perform such full-scope performance testing early in the development life cycle. To conduct performance testing early in the life cycle requires a more agile, explorative process.
- For the large scale enterprise performance testing projects, there is a bigger challenge in the scripting where scripting/design and execution happen without learning the system behavior. Running a script with baseline user load which does not yield any errors is not sufficient enough to say those can be executed for all the planned cycles. Does executing all the planned scripts give an assurance that the system works correctly just because the scripts were executed without errors, though performance tests were declared successful and the results were reported to management?
- Running all scripts simultaneously makes it difficult to tune and troubleshoot. Sometimes the performance test engineers need to go back and find exactly which area of the system is causing problems which are imperative. Hence, tuning and performance trouble shooting are not just the sequential but more of an iterative process.
- Running a single large test, or even several large tests, provides minimal information about system behavior. It means that, it doesn’t tell you the system behavior and relationship between...
the workload and system behavior. So the performance test engineers need to knowhow stable
the system would be and even if the NFRs and SLAs are meeting, how consistent the results
would be if the real workload varied. This may not make sure that the other areas of the system
are performing up to the SLA’s. It just means that the performance engineers will probably do lot
of extra work and end up circling back to the same point, performance tuning and troubleshooting
much later in the cycles.

• With or without NFRs and SLAs, traditional model performance testing activities are not sufficient
enough to guarantee for full coverage of performance testing.

On the other hand, the software development models are evolving in a fast-paced mode, where many of
the IT implementations are inclined to produce high quality software in less time. Now the question is
whether the current performance test practices enough to achieve the objectives in the revived
development landscape and a total paradigm shift in development models. Many organizations are
moving away from traditional waterfall model to agile methods, typically performance testing
happens in every sprint and repeated at the end of the product development. This raises many questions
such as, how do we test for performance within sprint after sprint? How efficient is it to conduct
performance testing again at the end of the product development? And is that sufficient enough to say
the system performance is better? But no clear answers. Every agile team tries to adopt their own
approach for performance testing, depending on the type of product, team, project timelines,
stakeholders, organization, etc., and even some don’t consider performance activity in their plans. As per
the agile manifesto, the definition of done is delivering the product. If performance testing is part of the
“Definition of Done”, then that testing can happen only during the last few days of the sprint cycle which
results in some challenges as well.

• Performance testing is not completely done at the system level, integrating all the features
developed in the sprint.
• If feature development continues across sprints, then they aren’t included in the performance
testing of the same sprint and not all the features developed in the sprint are tested.
• Not able to learn the system behavior and the performance team not being aware of multiple
dimensions of the system under test, to perform the planned tests and also think of the system
scalability.

Proposed Model – Risk-based exploratory performance testing

“Risk-based Exploratory Performance testing” can bail you out in such a tough situations. Does this mean
that you need to ignore traditional or agile performance testing practices in favor of exploratory tests?
Answer is “Not at all..!” This is an idea of combining exploratory performance testing techniques and
tactics in conjunction with regular performance testing.

This approach in functional testing heavily goes after one particular area of the system which is not part of
the scripted testing. To quote an example for it is, discovering system functional behavior when a form is
submitted for several times to the server by hitting the refresh multiple times. But in this concept of
exploratory performance, tests will be selected based on the risk associated with performance quality
parameters along the regular load or performance tests. This means, it allows us to learn the system as it
progresses in each of the performance cycles and subsequently fine tune the select criteria for
exploratory test selections until the high risk areas are covered. This will lead to a great deal of performance testing coverage and gives assurance on the below parameters.

- Speed - does the system respond quickly enough
- Capacity - is the infrastructure sized adequately
- Scalability - can the system grow to handle future volumes
- Stability - does the system behave correctly under load

Performance testing of a service or product when risk analysis indicates that failing in any of the above categories would be more costly than performing the tests. The proposed model is base-lined as a practice through which we can apply this concept across the application development life cycle models:

**Performance testing in waterfall implementations**

The diagram shows the traditional performance testing life cycle where the cycle starts with NFR (Non-Functional Requirements) analysis and ends with test results analysis, recommendations. During requirement analysis phase non-functional requirements are identified and gathered from the clients, which includes SLA’s, workload, volume of data, duration, application architecture, hardware etc.

Once the NFRs are finalized, planning and design of performance tests are done using the data provided by the client in the previous phase. During this phase we define the types of tests to be conducted, scenario objectives, and metrics to be measured and captured, data required, etc. As per the designed tests and scenarios, the environment needs to be set up and configured. Generally, Performance test engineers prefer an environment dedicated to their testing where they have complete control.

Next phase involves the development of test scripts and test assets that will be used in execution of performance tests.

During test execution the scenarios planned and created in previous phase are executed using the test assets developed. Monitors are set up in the test environments to measure the required metrics defined.

The output generated after the execution is used for analysis and report creation. During analysis phase, results and logs are analyzed to identify any performance bottleneck and issue. If the results do not meet SLAs then performance tuning of the system will be done and after tuning the test execution will be repeated to test the fix. The process continues until the SLAs are met for the NFRs received.
Performance testing in agile implementations

Agile Performance testing has to be in sync with the continuous sprints of the project. Since functionality is not available at once in agile projects, creation and modification of assets is required after every sprint. Whereas in waterfall model, performance testing is generally done parallel to user acceptance testing when the functionality is completely available and stable.

Like in Traditional performance cycle, in agile projects NFR analysis and configuration can be completed in the beginning of the cycle. The next phases that include Identify & coordinate tasks, develop test assets and execute tasks, analyze results and reports will be repeated after every sprint.

Usually the performance testing is done after completion of one print and then follows respective sprints as the project moves ahead. In stabilization phase when multiple fixes and frequent build are common performance testing becomes critical and difficult. It requires a very good coordination between performance analysts and developers.

It is also not necessary to test each and every build released by the development team. Performance testers should enquire with developers about the build where they can test performance fixes and optimizations.

As all the functionality is not available in agile at ones, stubs are required sometimes to enable performance testing for a particular functionality.
Risk-Based Exploratory performance testing is an approach subjecting to a system, application or tier/component to a set of unusual parameters or scenarios that are not represented from the business context when defining the NFRs or SLAs. But these parameters are nevertheless possible to break a system or impact the performance of overall system. This proposed exploratory concept can be viewed as an interactive process of simultaneous learning, developing test assets, test execution and analysis. Most often, these tests are designed by adjusting the approaches as the performance engineers learn more about the system behavior.

Exploratory performance testing is most useful when no defined SLAs or NFRs, and haven’t had implemented a statistical usage profile. The flow depicts how exploratory performance test can be included within the core performance life cycle. In the exploratory cycle, it uses an iterative approach where an exploratory selection criterion evolves as the performance engineer learns more about the system behavior or how the system has been tuned for a particular situation. Once the performance engineer finds a bug and after it gets fixed, the performance engineer can fine tune the exploratory tests to avoiding testing the same bug again. This way you can find additional problems in the remaining areas in the system. To automate the exploratory tests, choose whichever tool will get you started fastest, whether it could be a commercial tool or an open source tool with collection of existing reusable libraries.

The core performance test activities listed in the figure are most commonly occur across any successful performance testing projects, the key point to effectively implement all the activities is not when you conduct them, whether or not they overlap, or not cycle iteration pattern among them, but rather you understand and carefully consider concepts, applying them in a manner that is most valuable to your project. In this context “exploratory test activities” come into the flow as shown in the figure, is the phase where unplanned tests (exploratory test scenarios) will be researched and selected to detect or explore the system with a minimum guarantee of tuned system. The result of “Exploratory performance test cycle” phase leads the project back to “Develop test assets” where “exploratory tests will be created and documented along with the NFR /SLA based scenario implementations.

To implement in a Risk Based approach, the risk assessment has to be performed which is complex and is challenging to perform with complete objectivity. It is an assessment based on a consensus of opinion. It is therefore important for the risk assessment not to be carried out by the Testing Strategist/Project
Manager (PM)/Performance Test Lead alone. A large number of people involved in the project should contribute e.g. business owner, operational owner, users, development team, and live support and so on. This not only increases the quality of the assessment, but it also has the advantage that the different parties are more aware of the risks and the extent to which testing contributes to making these risks manageable. The critical functional areas will turn out to be the areas which have higher risk associated with them and need more attention during the test planning and execution.

Risk Based Exploratory Testing Framework

The risk based exploratory performance testing framework consists of the following four elements. These elements define and drive the framework.

Framework Elements:
A. Test Stages

A test stage is a period of time in the test life cycle during which the components of a software product are evaluated with a specific objective to determine whether or not requirements have been satisfied.

The common test stages in performance test life cycle are:

- Load Testing
- Soak/Endurance Testing
- Stress Testing
- Volume Testing

B. Performance Quality Characteristics

Performance Quality Characteristics are different aspects of system that can be used to measure or quantify a software system. These Characteristics can also be used to compare two or more systems.

The common performance quality characteristics considered for software systems are:

- User satisfaction
- Synchronicity
- Service Level Agreement (SLA) violation
- Response time trend
- Configuration
- Consistency
- Capacity
- Volume
- Optimization
- Efficiency
- Future growth
- Resource consumption
- Hardware / environment
- Reliability
- Robustness
- Failure mode
- Slow leak
- Recovery
- Data accuracy and security
- Interfaces
C. **Sub-Systems:**

A sub-system may be defined as a logical grouping of relative/dependent business scenarios which when put together meets a business need. This activity mainly focuses on determine what subsystem or component or tier of the application is the cause of the detected performance degrade or determine what are all the likely components can degrade the system performance. It is critical to involve the technical experts in these discussions.

The intent of component evaluation is to collect the information as a whole, capturing the system architecture to assess the specific areas of the system exhibiting performance issues. Evaluating the system components/tiers includes, but is not limited to, the following activities:

- Understand and capture the physical and logical architecture
- Identify any other process/systems using the architecture.
- When testing a web farm, consider the usage of the IP (internet protocol) switching techniques.
- Most times application servers and web servers acts as multi-homed servers consider the fact how these characteristics can have significant impact on performance of the system.

Some examples of an eCommerce implementation could be:

- Customer Relationship Management
- Portal
- Ordering System
- Manufacturing System
- Inventory System
- Content Management System

In client specific custom solutions, the key components including 3rd party system may be considered as sub-systems.

D. **Relative Importance**

Relative importance of sub-systems will depend on “Frequency of Use” and “Chance of fault occurrence” in those sub-systems.

The test strategist/test lead/Project Manager will have to specify the “Frequency of use” to be either “High/Medium/Low” based on the number of actions/transactions a user will perform using the sub-system in a day. This can be determined either based on the business requirements or inputs of business owner, users and technical experts of the sub-system/solution/project.

The test strategist/test lead/Project Manager will have to specify “Chance of Fault Occurrence” to be either “High/Medium/Low” based on the previous experience of implementing the sub-system or complexity involved or amount of configuration involved etc.

The “Chance of Fault Occurrence” is the critical parameter of the two and it may be derived or may have dependencies with:

- Design complexity of the Sub-System or Business Scenario
- Customizations involved in Sub-System or Business Scenario
- Amount of Configuration in Sub-System or Business Scenario
On the inputs for the two parameters (i.e. Frequency of use and Chance of Fault Occurrence”) scalar product will be applied based on the following scale.

<table>
<thead>
<tr>
<th>“Frequency of use” Vs.</th>
<th>High</th>
<th>Medium</th>
<th>Low</th>
</tr>
</thead>
<tbody>
<tr>
<td>“Chance of Fault Occurrence”</td>
<td>High</td>
<td>High</td>
<td>Medium</td>
</tr>
<tr>
<td>Medium</td>
<td>High</td>
<td>Medium</td>
<td>Low</td>
</tr>
<tr>
<td>Low</td>
<td>Medium</td>
<td>Low</td>
<td>Low</td>
</tr>
</tbody>
</table>

The determination of relative importance should preferably be done with inputs from all stakeholders or at a minimum test strategist should determine the relative importance and get it reviewed by stakeholders. This will ensure that a general image of the importance of the sub-system is obtained from all concerned parties.

**Test Stage**
Risk based testing assessment can be performed for all the performance test stage within a project or may be performed independently for each stage. This determination will have to be made by test strategist in the beginning by discussing with business owner, operational support expert, technical expert etc.

Once a test stage for which risk based testing assessment is determined the next step is to review and finalize the quality characteristics which will play a key role in the test stage which is described in detail in the next section.

**Select Performance Quality Characteristics**
A selection of performance quality characteristics on which the tests must focus and include, with the help of the Business, Operational support, Technical support and other parties involved e.g. Service
Management. These quality characteristics may be used for reporting to all the stakeholders during test execution and completion.

Some characteristics are difficult to test, particularly if requirements are not clear. A substantial part of the effort should be devoted to determine the relevant quality requirements as measurably and unambiguously as possible.

Some quality characteristics demand more testing effort than others. Since a testing strategy that cannot be fulfilled is of no use, an estimate must be made for each selected performance quality characteristic. The relationship is established based on the scope of the testing stage and the objective of the testing project.

The significance of a Performance Quality characteristic may be dependent on the test stage. The table below is an example of selection of Quality Characteristics for different test stages:

<table>
<thead>
<tr>
<th>Quality Characteristic Vs. Test Stage</th>
<th>Component</th>
<th>Endurance</th>
<th>Investigation</th>
<th>Load</th>
<th>Stress</th>
<th>Validation</th>
</tr>
</thead>
<tbody>
<tr>
<td>User satisfaction</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Synchronicity</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>SLA violation</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Response time trend</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Consistency</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Capacity</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Volume</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Efficiency</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Future growth</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Resource consumption</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Reliability</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Robustness</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Failure mode</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Slow leak</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Recovery</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>

**Select Sub-Systems**

Every system under test consideration can be divided into subsystems. Each subsystem may have different levels of risk for the client / project / business. In principle, the division into subsystems may be the same as described in the system architecture.

In some projects there may be an overriding reason for making deviations from system architecture. If there is a deviation from system architecture, there must be clear indication for the motivation for this. One of the reasons why deviation is made could be because of the order in which development is building the system and delivering it to testing and later deploying it.

If the system only consists of one component/product then key features or functionalities of the product/component may be considered as sub-systems for risk assessment.
The sub-systems which are relevant in the current project should be selected for inclusion in the risk assessment. The identified sub systems will become one of the key components of the risk assessment/risk based testing approach as these sub systems will aid in uncovering the inherent risks in the product/component. It is recommended that the identification, selection and inclusion of sub-systems is performed with (or in consultation with) business owners, technical owners and users.

<table>
<thead>
<tr>
<th>Sub-System</th>
<th>In Scope (Yes/No)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Customer Relationship Management</td>
<td>Yes</td>
</tr>
<tr>
<td>Portal</td>
<td>Yes</td>
</tr>
<tr>
<td>Order Management</td>
<td>Yes</td>
</tr>
<tr>
<td>Manufacturing System</td>
<td>Yes</td>
</tr>
<tr>
<td>Inventory Management</td>
<td>Yes</td>
</tr>
<tr>
<td>Content Management System</td>
<td>Yes</td>
</tr>
</tbody>
</table>

### Determine Relative Importance of Sub-System

Once sub-systems are identified determination of relative importance of the sub-systems will have to be done. The accurate relative importance of subsystems can only be determined if a good test strategist has understanding of system and business. The importance may be adjusted when more information is available and a detailed risk assessment performed.

An example of relative importance of sub-systems of Portal Implementation project:

<table>
<thead>
<tr>
<th>Sub-System</th>
<th>Frequency of Use</th>
<th>Chance of Fault Occurrence</th>
<th>Relative Importance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Customer Relationship Management</td>
<td>High</td>
<td>High</td>
<td>High</td>
</tr>
<tr>
<td>Portal</td>
<td>High</td>
<td>High</td>
<td>High</td>
</tr>
<tr>
<td>Order Management</td>
<td>High</td>
<td>High</td>
<td>High</td>
</tr>
<tr>
<td>Manufacturing System</td>
<td>High</td>
<td>Medium</td>
<td>High</td>
</tr>
<tr>
<td>Inventory Management</td>
<td>Low</td>
<td>Medium</td>
<td>Low</td>
</tr>
<tr>
<td>Content Management System</td>
<td>Low</td>
<td>Medium</td>
<td>Low</td>
</tr>
</tbody>
</table>

### Determine Sub-System Priority

In the previous steps quality characteristics and sub systems were identified and relative importance of Sub-System determined.

The relative importance of the sub-system denotes importance of sub-system relative to other sub-system in the solution/product/system. For accurate risk assessment it is critical that each sub-system is
analyzed for its role in the solution/product/system based on the quality characteristics rather than only on their relative importance.

By analyzing each sub-system using the quality characteristics the sub-systems are assessed based on their role in the overall project and based on their position in the solution. For each sub-system every quality characteristic is assessed and relative importance of the quality characteristic for the sub-system is specified. This enables test strategist to distinctly specify the significance of each quality characteristic for every sub-system for e.g. performance is important, but this holds predominantly for Customer Relationship Management (maybe this is front-end access to the system) and not necessarily for Inventory etc.

The steps to determine the Sub-System priority is to:

- Select relative importance of every quality characteristic for every sub-system in scope of the risk assessment. For example relative importance of a quality characteristic may be high for sub-system 1 and low for sub-system 2.
- Once relative importance of quality characteristics is determined for all the quality characteristics for the sub-system, the scalar product of relative importance of quality characteristics and relative importance of Sub-system determines the overall “Sub-System Priority”

<table>
<thead>
<tr>
<th>Sub-System</th>
<th>Quality Characteristic</th>
<th>Relative Importance of Quality Characteristic</th>
<th>Relative Importance of Sub System</th>
<th>Sub-System Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>Customer Relationship Management</td>
<td>User satisfaction</td>
<td>Low</td>
<td>High</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Synchronicity</td>
<td>High</td>
<td>High</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SLA violation</td>
<td>High</td>
<td>High</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Response time trend</td>
<td>Low</td>
<td>High</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Consistency</td>
<td>Low</td>
<td>High</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Capacity</td>
<td>Medium</td>
<td>High</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Volume</td>
<td>Low</td>
<td>High</td>
<td></td>
</tr>
</tbody>
</table>

**Identify Business Scenarios**

In the previous steps sub-system priority was determined. The sub-system priority denotes the importance of the sub-system from product/solution/system perspective. The business priority denotes the importance of the sub-system from business perspective.

A sub-system may also be defined as a logical grouping of relative/dependent business scenarios which when put together meets a business need. The business scenarios in a sub-system are basic yet key operations which have significant impact on the day to day business operations of the user.

To determine the business priority of the sub-system key business scenarios within the sub-system will have to be identified. The day to day business activities or key business activities will be the basis for identifying the key business scenarios with in the sub-system. Inputs from business owners, users, system administrators, technical owners and business analysts are very critical in identifying the correct business scenarios. Alternatively test strategist may identify the business scenarios and get them reviewed by all the stakeholders.
The design of the system may also be used to derive the business scenarios. If design work is in progress, it is suggested that business scenarios be revisited as design gets finalized or changed.

An example of business scenarios for sub-systems could be:

<table>
<thead>
<tr>
<th>Sub-System</th>
<th>Business Scenario</th>
</tr>
</thead>
<tbody>
<tr>
<td>Customer Relationship Management</td>
<td>Customer Creation</td>
</tr>
<tr>
<td></td>
<td>Account Creation</td>
</tr>
<tr>
<td></td>
<td>Selection of Products &amp;Services</td>
</tr>
<tr>
<td></td>
<td>Resource Management</td>
</tr>
<tr>
<td></td>
<td>Payment Method/CC change</td>
</tr>
<tr>
<td></td>
<td>Payments/Refunds/Adjust</td>
</tr>
</tbody>
</table>

**Determine Business Risks (Business Priority)**
Once business scenarios for a sub-system are identified determination of business risks for those scenarios will have to be done. The business risks of business scenarios will denote the business priority.

The business risks can be determined only if significance of the business scenarios in a real time business scenario is known in terms of “Probability of Failure” and their “Impact”. The scalar product of “Probability of Failure” and “Impact” will be Business Risk (Business Priority) associated with the Business Scenario.

Determination of business risk should be performed by inputs or from Business owner, operational owner and other parties involved or test strategist should identify the business risks based on his understanding of the business and get it reviewed and approved by all stakeholders.

<table>
<thead>
<tr>
<th>Sub-System</th>
<th>Business Scenario</th>
<th>Probability of Failure</th>
<th>Impact</th>
<th>Business Risk (Business Priority)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Customer Relationship Management</td>
<td>Customer Creation</td>
<td>High</td>
<td>Low</td>
<td>Medium</td>
</tr>
<tr>
<td></td>
<td>Account Creation</td>
<td>Low</td>
<td>High</td>
<td>Medium</td>
</tr>
<tr>
<td></td>
<td>Selection of Products &amp;Services</td>
<td>Medium</td>
<td>High</td>
<td>High</td>
</tr>
<tr>
<td></td>
<td>Resource Management</td>
<td>High</td>
<td>Medium</td>
<td>High</td>
</tr>
<tr>
<td></td>
<td>Payment Method/CC change</td>
<td>Medium</td>
<td>Medium</td>
<td>Medium</td>
</tr>
<tr>
<td></td>
<td>Payments/Refunds/Adjust</td>
<td>High</td>
<td>Low</td>
<td>Medium</td>
</tr>
</tbody>
</table>

**Determine Test Priority**
Having determined the sub-system priority which denotes the importance of sub-system from product perspective and determined business risks of business scenarios which denotes significance of business scenarios from business or operational perspective, the next step is to determine the overall “Test Priority”.

14
The “Test Priority” associated with business scenarios will be scalar product of sub-system priority and business risk (priority). The determination of test priority at business scenario level helps identify risks of the product/system/solution at the lowest possible level thus projecting a realistic picture of associated risks. The higher the test priority for a sub-system higher should the focus on the sub-system during the planning phase.

<table>
<thead>
<tr>
<th>Sub-System</th>
<th>Business Scenario</th>
<th>Business (Priority)</th>
<th>Risk</th>
<th>Product Priority</th>
<th>Test Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>Customer Relationship Management</td>
<td>Customer Creation</td>
<td>Medium</td>
<td></td>
<td></td>
<td>High</td>
</tr>
<tr>
<td></td>
<td>Account Creation</td>
<td>Medium</td>
<td></td>
<td></td>
<td>High</td>
</tr>
<tr>
<td></td>
<td>Selection of Products &amp; Services</td>
<td>High</td>
<td>High</td>
<td>Critical</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Resource Management</td>
<td>High</td>
<td>Critical</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Bill Cycle Change</td>
<td>Medium</td>
<td>High</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Payment Method/CC change</td>
<td>Medium</td>
<td>High</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Payments/Refunds/Adjust</td>
<td>Medium</td>
<td>High</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Exploratory performance test selection**

The absence of clear NFRs and SLAs works as a boon in the concept of “Exploratory Performance Testing”. It gives an opportunity to explore more combinations that might not be uncovered in case of normal approach. It drives you to search for possibilities in each and every aspect of performance engineering.

The concept divides the exploration into different aspects and which gives you the accurate baseline performance scenario saving us a lot of time and effort. The following aspects lead you to drive the conclusion in the end.

- Exploratory for Load
- Exploratory for Volume
- Exploratory for System Resources
- Exploratory for Functional Combinations

**Exploratory for Load**

In case of unclear NFRs and SLAs the load testing becomes an ambiguous task. It becomes difficult to decide the correct load on the system and baseline it. Exploration for load is more practical than theory. To find the correct load we need to try to stress the system with stress testing. Once we have the breakdown point, we can baseline the load for performance testing near this point.

The results of the Performance Tests will give a hint of the bearing capacity of the application in terms of load. This in turn will help us to recommend the current load bearing capacity to the client. Based on these results client can ask for further optimization.
Exploratory for volume

To find the correct set of data to do a performance test is a challenge if the requirements are not defined accurately. To overcome such difficult situation we can do some small volume test of the application to know the point after which the responses of transactions start increasing in an exponential manner. The point defines the volume that should be made as baseline for further tests. The results should be sent as recommendations for optimization if required.

Exploratory for System Resources

System resources play an important part in the performance of an application. The system resources in different servers in an application can define the performance pattern of the application. The approach in this aspect is a bit different than the previous two aspects. Here we will take help from Industry standards to set up the first set of infrastructure. This point onwards we will monitor the performance of the main system resources during a performance tests. The monitoring will use the usual monitoring tools in performance testing. The results will lead us to a conclusion whether the standard system resources are enough for the kind of performance testing or we need to boost the infrastructure.
Exploratory for Functional Combinations

This is the toughest part of this concept and involves lot of functional knowledge. The functional knowledge will enable you think more scenarios involving potential performance issues. Depending on the application such scenarios can be executed using a tool or manually. Such permutation and combination ensures the coverage of the periphery of the target functionality and can uncover potential performance issues.

Develop Test Assets and Execution

After exploratory test selection, we need to work on test assets. Before creating assets it is critical to identify the right tools as per your requirements. Asset creation is usually a onetime process in projects following waterfall approach. But in agile projects, where functionality is not available at once, test assets are created or modified after each sprint. Test assets include scenarios, test cases, scripts, templates etc.

Benefits

The following are the benefits for Risk-based exploratory performance testing approach:

- Focus on the high priority business scenarios
- The exploration approach covers all the aspects of the performance testing and is more effective in finding the potential issues.
- No NFRs and SLAs have to be defined for this approach that gives an opening to unlimited opportunities.
- Develop performance testing strategy/test plan and performance test scenarios focused on these areas
• Helps in developing a better exit criteria for testing thus helping management to focus more on critical or high scenarios of the project
• Map performance scenarios back to risk/quality characteristic (traceability)
• Plan test execution based on the priorities
• Track test execution by risk/quality characteristic
• Report based on quality characteristics and test priorities and take timely measures and decisions based on residual risk

Conclusion
This technique may take some time initially to do all the brainstorming but it surely reduces and a lot of risks by finding additional exploratory test cases. More and more performance issues can be found if we use this method effectively.
People may have their own opinions regarding this concept but they all will agree that word “Risk based exploratory” is now knocking at the door of performance testing world. Let us all help each other in opening that door and embrace this revolutionary change.

References
• Performance testing guidance for web applications byJ.D. Meier, Carlos Farre, Prashant Bansode, Scott Barber, Dennis Rea
• www.perftestplus.com
• Exploring Exploratory Testing by Andy Tinkham (FIT) and CemKaner (FIT)
• The Nature of Exploratory Testing by CemKaner and James Bach