Quantify Perceived Performance

"Quantifying and rating the performance
of (virtual) applications"

Ingmar Verheij
Senior Consultant
PepperByte

Date September 20, 2011
Document version 1.0
Status Definitive

GENERAL

General Information

Organization	PepperByte B.V.
Address	Postbus 1032
Postcode and Town	2240 BA Wassenaar
Telephone no.	+31 88 7890000
Fax no.	+31 88 7890010
E-mail address author	i.verheij@pepperbyte.com
General e-mail address	info@pepperbyte.com
CoC no.	27243490
VAT no.	NL 813379313-B01

Version management

Version	Date	Type of amendment	Author	Reviewer	Release
0.1	15-07-11	Initial, research	I. Verheij	-	-
0.2	18-07-11	User actions, process	I. Verheij	-	-
0.3	19-07-11	Process, references	I. Verheij	-	-
0.4	20-07-11	Examples added	I. Verheij	Daniel Nikolic, Jeroen Tielen, Kees Baggerman, Okke Garling, Andrew Wood	Private
0.5	29-07-11	Comments from reviewers processed	I. Verheij	Daniel Nikolic, Kees Baggerman, Okke Garling, Andrew Wood, Tim Mangan	Private
0.6	26-08-11	Comments from reviewers processed	I. Verheij	Daniel Nikolic, Jeroen Tielen, Marco Hokke, Kees Baggerman, Okke Garling, Andrew Wood, Tim Mangan	Private
1.0	20-09-11	Final version	I. Verheij	-	Public

http://www.pepperbyte.com/

TABLE OF CONTENTS

1 Summary

Determining the user experience is challenging since the experience is based on many factors. The performance of an application is one of the factors that determine the user experience. If the performance of an application is bad, the user will grade the experience as bad.

The performance of an application can be influenced by many components. Delivering the application to a user usually involves a rather complex chain consisting of many links. For instance a (virtual) desktop system that has to compete for resources, a WAN connection that experiences delay and jitter or a backend infrastructure that has to deliver content. Reduced services in any component has the same effect for the end user – their perception of the quality of the performance is reduced. The user doesn't want to know what the cause is (although it helps to understand and maybe accept) they only perceive a decrease in performance.

Quantifying and grading the performance of an application as perceived by a user is challenging and requires guidelines and thresholds. I've done research about the psychological effect of (variable) response times on the user experience and existing methods of grading performance.

In this document I will explain a process to quantify and rate the perceived performance of an application based on the response time of an application. For this purpose two new indexes are introduced: Application Responsiveness Index (ARI) and Perceived Performance Index (PPI).

Ingmar Verheij
Senior Consultant
PepperByte BV

Wassenaar, September 2011

2 How to Quantify Perceived Performance

This document describes how you can quantify, rate and compare the performance of an application as it is perceived by a user. By simulating user actions and measuring the response times it is possible to compute a set of values that give you a reliable indication of perceived performance. That said, the method is not intended to capture the user experience as a discrete number.

The goal of the process is to allow any application to be rated, independently from the delivery service. Whether the application is natively installed or virtualized, delivered via a virtual or physical desktop (SBC, VDI or fat client) or the operating system used, the process of quantifying and rating the perceived performance remains the same. The process is intended to give an indication of how an average user will rate the responsiveness of an application. By comparing the results across delivery services one may make an informed decision on which will give the best user experience.

The process is designed to quickly rate an application using a point measurement. The measurement (and rating) can be repeated over time to get an insight in the trend (for instance during peak hours).

2.1 ARI – Application Responsiveness Index

The Application Responsiveness Index (or ARI) is a result based on the level of responsiveness of the application. Based on the outcome of the ARI an application can be rated with a predefined set of qualifiers: e.g. "Excellent", "Good" or "Poor". The ARI can be used as an indication for the level of responsiveness perceived by users at a given time.

2.2 PPI – Perceived Performance Index

An addition to the ARI is the Perceived Performance Index (or PPI). The PPI doesn't just rate the response time of the application but includes the variability of the response time. This means that if there is a great fluctuation in the response time the score will be lower resulting in less rating. The PPI is a good indication of the perceived performance of an application over time.

3 Research

The process of quantifying and rating the perceived performance is based on the distillation of a number of published works about user experience, response times and perceived performance.

The terminology "perceived performance" was first described in a white paper by Tim Mangan in 2003 [1]. Tim Mangan described a different approach of looking at performance, focusing on the end-user instead of a system. This is described in chapter 3.1.

Finding thresholds for response times in user machine interaction has been extensively researched. There are numerous documents published, often with conflicting findings. Usually, the result of the studies consists of the average result of a certain representative group. However, these papers are difficult to read and interpret, and do not always give enough clarity about the result of the study. Therefore, it was necessary to read multiple papers, combine the results and then validate the process described in this document. The results can be read in chapter 3.2.

The numbers in a dataset are calculated using certain mathematical rules. The mathematics used are described in chapter 3.3.

3.1 Perceived Performance

Tim Mangan is a Microsoft Most Valuable Professional (MVP) for Virtualization (App-V). The MVP Program is how Microsoft recognizes and awards professionals in the field who make a significant community contribution. Tim Mangan is also a Citrix Technology Professional (CTP). The CTP is how Citrix recognizes top "thought leaders" in the industry.

Tim Mangan has written multiple papers about perceived performance in a virtual desktop environment. He clearly explains the difference between computational performance and perceived performance.

> *"Computational capacity is the total amount of useful work that can be accomplished on a system in a given fixed period of time"*
> Tim Mangan, Perceived Performance, September 18, 2003 [1]

> *"A methodology where one analyzes the system with a goal of improving user productivity by focusing on issues that affect the performance as perceived by the users"*
> Tim Mangan, Perceived Performance Reloaded, May 2011 [2]

Mangan recommends changing the focus from *computational performance*, which is the traditional way of looking at performance, to *perceived performance*. Although computational performance can give us insight of which link in the chain might be causing the performance decrease, there is no guarantee that if all items report an acceptable value the perceived performance is acceptable.

Unfortunately there is no information about thresholds in the computational performance measured numbers, or how they relate to the perceived performance of user actions. Without thresholds it is impossible to rate a result, making it less valuable.

3.2 Psychological Research

Much research has been done to determine how users react when the time between an action and the result of that action (response time) is increased or fluctuates. The effects studied are user experience, error rate and productivity. The vast majority of humans like a consistent speed which is not too fast and not too slow. If the system is too fast the chance of errors will increase, lowering productivity; if the system is too slow it will decrease user experience and productivity. More variability in response time could impact multiple aspects although there is no consistent outcome of that.

Most interesting is that a human is highly adaptable and will learn to work with the speed given, decreasing error rate and increasing productivity.

In this section, we'll highlight a number of the more important research papers in this field.

Response time in Man-Computer Conversational Transactions
Author Robert B. Miller

Most research done about response times and the impact on the user experience or user acceptance refer to a research done in 1968 by Robert B. Miller. This paper "Response time in Man-Computer Conversational Transactions" [3] presented research that had been conducted to find the thresholds for response times for certain actions between a human and a computer.

The content of the research is used to determine the thresholds of three categories. The categories are described in detail in chapter 4.2. They form the base of the process of quantifying perceived performance.

Response Time and Display Rate in Human Performance with Computers
Author Ben Schneiderman.

This research was conducted by Ben Schneiderman to study the effects of response time and display rate in human performance with computers. The results of this study are published in 1984 in the paper "Response Time and Display Rate in Human Performance with Computers" [4].

The research included the effect of variability in response time on the user experience, error rate and productivity. There is no clear evidence that variability in the response time negatively affects any of the effects over a long period. The effects seems to be lifted because humans adept. However, there are some effects measured, especially when the response time is much higher (or lower) than expected.

The effects of variability, although lifted over a longer period, and the measured effect of high response times form the base of the Perceived Performance Index (PPI). A penalty is added to the Application Responsiveness Index (ARI) based on the spreading of the numbers in the data-set captured.

Usability Engineering
Author Jakob Nielsen

In 1993 the book "Usability Engineering" by Jakob Nielsen discussed the ease of using a graphical interface. In chapter 5, *Usability Heuristics [5]*, three categories are described based on the research of Robert B. Miller in 1968.

Excerpt from Chapter 5 in the book "Usability Engineering"
0.1 second: is about the limit for having the user feel that the system is reacting instantaneously, meaning that no special feedback is necessary except to display the result. *100 milliseconds is also the norm being used for an average that users can perceive.*

1.0 second: is about the limit for the user's flow of thought to stay uninterrupted, even though the user will notice the delay. Normally, no special feedback is necessary during delays of more than 0.1 but less than 1.0 second, but the user does lose the feeling of operating directly on the data.

10 seconds: is about the limit for keeping the user's attention focused on the dialogue. For longer delays, users will want to perform other tasks while waiting for the computer to finish, so they should be given feedback indicating when the computer expects to be done. Feedback during the delay is especially important if the response time is likely to be highly variable, since users will then not know what to expect.

The three categories described match the results of the Robert B. Miller in 1968 and the categories used to classify user actions as described in chapter 4.2.

Is 100 Milliseconds Too Fast?
Author James R. Dabrowski and Ethan V. Munson

For a long time application developers used the 100 millisecond limit as a rule for building their interface. Every application needed to respond within 100 milliseconds or it would be perceived to be slow. In 2001 James R. Dabrowski and Ethan V. Munson did a research about this threshold and published the results in the paper "Is 100 Milliseconds Too Fast?" [6].

With this research Dabrowski and Munson attempted to determine if the norm of 100 milliseconds is valid or should be adjusted. In their research they examined five different actions in an application and determined when users perceived a delay (using the staircase method).

The result of the test, which can be seen in the table below, turned out to be between 150 and 200 milliseconds. The results match the thresholds as described by Robert B. Miller in 1968 [3].

	Menu	Button	Dialog	Typing	Form
Mean	171.98	197.56	192.98	156.07	146.81
Sd	64.68	68.26	54.74	52.63	46.22
S.E.	14.11	14.89	11.95	11.48	10.09
Min.	22.50	22.50	59.50	39.93	37.50
Max.	307.08	366.67	267.92	241.31	215.45

This result of the paper implicates that the 100 milliseconds is not a fixed value but rather a guideline. Rating a response time should therefore be based on a range of numbers instead of a fixed value.

Defining the Application Performance Index
Author Peter J. Sevcik

In 2005 Peter J. Sevcik published the article "Defining the Application Performance Index" [7]. The article describes three zones in which the speed of an application may be categorized, where each zone is separated by a threshold.

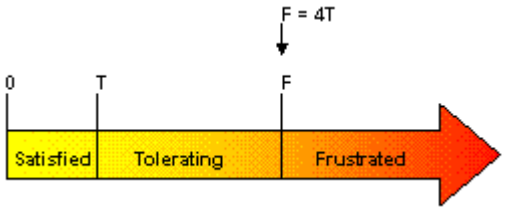


Figure 1 - Apdex zones

The threshold from the psychological research, as described above, matches the tolerated value: T, which divides the first two zones. The value where users get frustrated, and aborting the process, is the value F. This is the value that divides the second and the third zone. Research conducted by Peter J. Sevcik and his colleagues prove that F is a function of T, where F equals T multiplied by four (F=4T).

This rule is the basis of the Apdex method as defined by the Apdex Alliance.

The Apdex Alliance is a group of companies that are collaborating to establish the Apdex standard. These companies have perceived the need for a simple and uniform way to report on application performance, they are adopting the Apdex method in their internal operations or software products, and they are participating in the work of refining and extending the definition of the Apdex specifications. Alliance contributing members who incorporate the standard into their products may use the Apdex name or logo where the Alliance has certified them as compliant.

In January 2007, the Alliance comprised 11 contributing member companies, and over 200 individual members.

By using two thresholds instead of one a range of numbers is used to rate a response time. The result is rated on a scale determine by the Apdex Alliance, this is described in chapter 5.3.

3.3 Apdex

The Application Performance Index (Apdex) is "The industry standard in measuring application performance" according to the Apdex alliance. Using the Apdex it's possible to convert an absolute number, for instance a response time, to a relative number and rate the outcome. The Apdex is described in the *Apdex Technical Specification* [9].

A relative number is necessary to compare results with a different distribution. Comparing results that have a distribution between 1 and 10 with results that have a distribution between 1 and 100 is not possible without a relative number. By using the thresholds described in chapter 4.2 to create a relative number, the result can be used to rate the outcome.

Using Apdex, results are place into one of three zones. The zones are separated using the T and F where F equals 4T.

The zones are named:

- Satisfied;
- Tolerating;
- Frustrated.

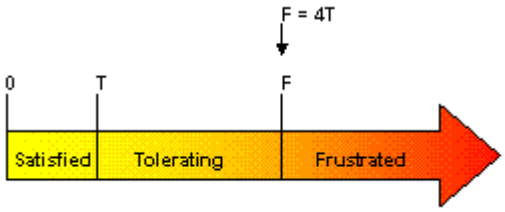


Figure 2 - Apdex zones

Using the Apdex formula a relative number is calculated between 1 and 0. The result is based on the number of results in a zone (satisfied, tolerating or frustrated) without considering the value.

$$Apdex_T = \frac{Satisfied\ count + \frac{Tolerating\ count}{2}}{Total\ samples}$$

Figure 3 - Apdex formula

The result is rated using the Apdex tower with a name (Excellent, Good, etc.) and a color (Blue, Green, etc.) that is associated with the name. The ratings are divided by an upper and a lower threshold.

Color	Name	Upper threshold	Lower threshold
Blue	Excellent	1,00	0,94
Green	Good	0,93	0,85
Yellow	Fair	0,84	0,70
Red	Poor	0,69	0,50
Gray	Unacceptable	0,49	0,00

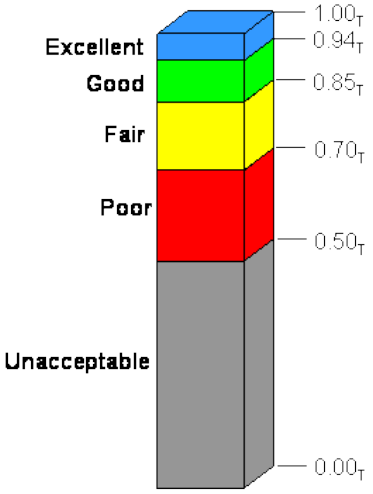


Figure 4 - Apdex rating tower

3.4 Mathematics

Results are calculated using standard statistical methods. Although well-known functions (like the average) are used, it is perhaps useful to recap some of the other functions.

The difference between mean and median
The average of a set of numbers can be determined with the **mean** and the **median**. To illustrate the difference between these two we'll take this set of 5 numbers:

4 - 2 - 14 - 2 - 3

Mean
The mean is the also known as the **arithmetic mean** and is calculated by adding N numbers in a data set together and dividing it by N. The mean of the 5 numbers above is 5.0

$$\frac{4 + 2 + 14 + 2 + 3}{5} = 5.0$$

Median

The median of a data set is the middle number of a set, sorted in numerical order. With an odd-numbered data set this is the number that is in the middle. When there is an even-numbered data set the mean of the two middle numbers is taken. The median of the 5 numbers above is 3.0

Odd-numbered

$$2 + 2 + \mathbf{3} + 4 + 14 = 3.0$$

Even-numbered

$$2 + 2 + \mathbf{3 + 4} + 5 + 14 = (3 + 4) / 2 = 3.5$$

Mean Absolute Deviation (MAD)

The Mean Absolute Deviation (MAD) [8] is the absolute difference between a central tendency, the mean average, and the elements in the dataset. The MAD is calculated by using the mean average of a subset, the central tendency, and for each value in the subset calculating the absolute difference with the mean average. The average of all values is the MAD.

To illustrate the how the MAD is calculated we'll take this set of 5 numbers:

$$2, 2, 3, 4, 14$$

The arithmetic mean of these 5 numbers is 5:

$$\frac{2 + 2 + 3 + 4 + 14}{5} = 5$$

For each value in the dataset the **absolute** distance to the central tendency is calculated:

$$2 - 5 = 3$$
$$2 - 5 = 3$$
$$3 - 5 = 2$$
$$4 - 5 = 1$$
$$14 - 5 = 9$$

The arithmetic mean of all values is the Mean Absolute Deviation:

$$\frac{3 + 3 + 2 + 1 + 9}{5} = 3.6$$

4 User actions

Users experience the performance of an application by performing actions and waiting for the result to appear on the screen. In order to quantify such performance, it is necessary to measure the amount of time from a user request to the system response.

In this chapter the necessity of automated testing is discussed, instead of measuring "real" users. The user actions simulated (just like normal users perform) are categorized and a threshold is set for each category. Each category is illustrated with an example showing screenshots of an application and the action performed.
Finally the learning effect that occurs due to optimization techniques is explained and prevented.

4.1 Automated Testing

Measuring the response time is best done by an automated system since such a system can regulate the actions. Simulating user actions can be done using scripting languages like AutoIT, vbScript, PowerShell, etc. but this only a part of the solution. Using scripting languages requires scripting skills and usually much time to setup (and optimize).

Load testing applications like 'Citrix EdgeSight for LoadTesting', 'HP Loadrunner' and 'DeNamiK LoadGen' are not only able to simulate user actions but are also able to initiate user sessions on a virtual (desktop) environment. By using a loadtest application, a fully automated test can be created.

The accuracy of the measurement depends on the interval the (automated) process is checking the screen for a certain result. For instance if a bitmap region on a screen should appear within 0,1 second an interval of 1 second (between each check) would not result in an accurate measurement. The accuracy (interval) should be around 10~15 milliseconds, preferably configurable in the application used.

On a virtual infrastructure the most accurate result, compared with the way users perceive results, is achieved when client side testing is used instead of server side testing. With server-side testing a process is launched on the server (the virtual host) to simulate user actions and measure the results. The downside of a process running on the server is that the effects of the delivery mechanism to the client (e.g. a remoting protocol over a LAN or WAN connection) are ignored. Since the script is dependent on the processing power of the server hosting the process, the measurements can be influenced by the load generated (which should be prevented). On a (very) busy virtualized server the measurement can be affected by a clock drift.

Client-side testing is done by using a process that is launched on the client and then launches a session to the server. The results of the actions are based on the same output as the users would see on their screen.

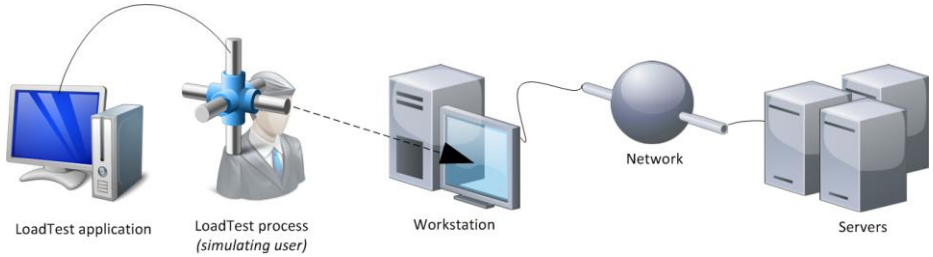


Figure 5- Client side testing with a simulated user

All three products mentioned are able to perform client-side testing.

4.2 Categories

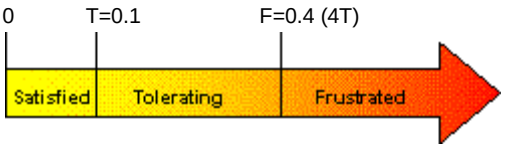
User actions can be categorized in three ways. By categorizing user actions it's possible to apply thresholds and compare the results between actions. Each category represents a type of action and is accompanied by a threshold, called T. The threshold is a fixed value based on the distillation of a number of published works (as described in chapter 3.2).

For determining the responsiveness of an application the process only uses categories A and B. Category C will be ignored since it measures processing speed instead of responsiveness.

For each category the measurement of the response time is started *after* the user action and stopped when the expected result is displayed on the screen. Each of the three categories is explained and accompanied by an example.

Category A **Acknowledgment of command**

Threshold *0.1 second*

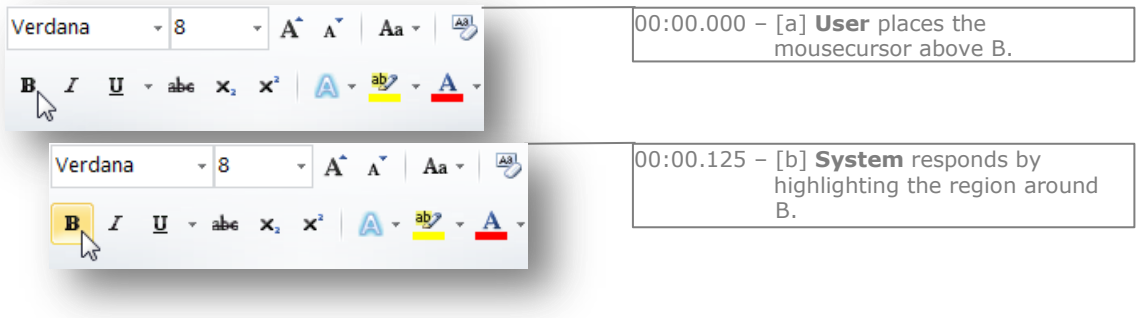


Figuur 1 : Thresholds in Apdex zones

Description

The time required to acknowledge the user that a command given by the user [a] is received by the system [b]. The acknowledgement is given by the system via visual feedback, which can be perceived by a user.

An example of a command [a] is pressing a key on a keyboard or clicking a button on a mouse, an example of a response is a change in color of a text or a change of the mouse cursor. With the response the user knows that the command is given and received by the system and that the system is responding.



As long as the response time stays within the threshold, the user has the feeling that the application responds fluently and isn't held back by the system.

Category B **Simple task**

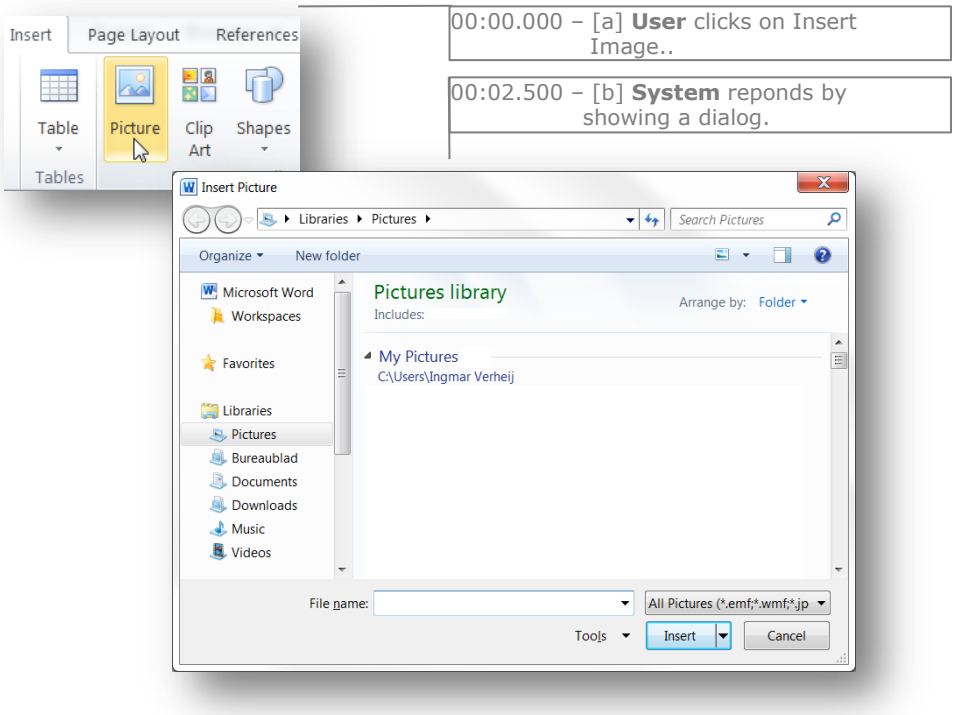
Threshold *1 second*

Figuur 2 - Thresholds in Apdex zones

Description

An action (as part of a sequence) where the system performs a simple task [a] and shows the system shows a response [b] keeping the state of user in a flow.

An example of a command [a] is clicking on a button followed by a dialog shown by the system [b]. After showing the dialog [b] the users will perform another action [a] creating a sequence of simple actions.



As long as the response time remains within the threshold the user won't get distracted and can continue executing the planned actions without thinking about the next step.

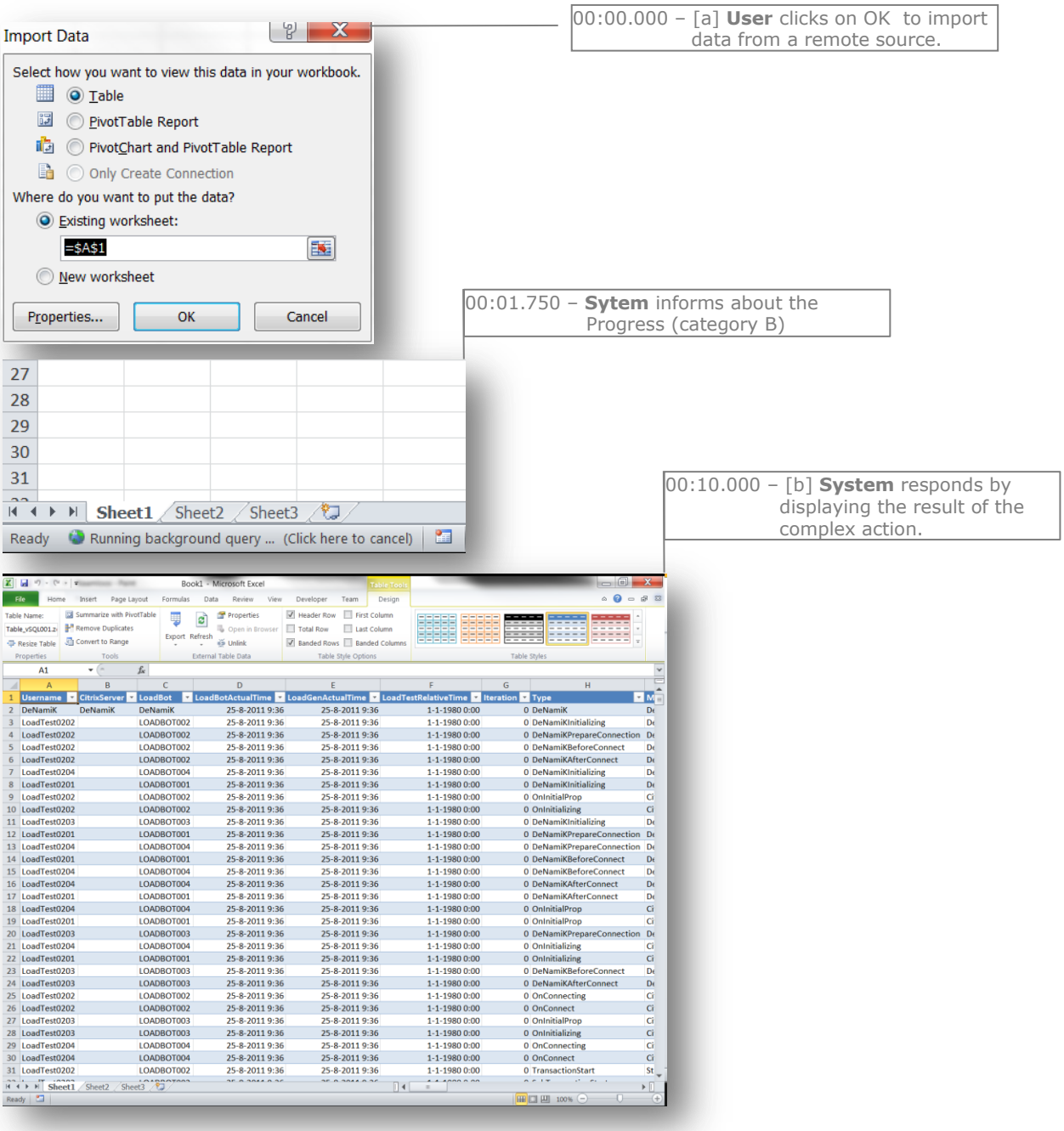
Category C **Complex task**

Threshold *10 seconds*

Figuur 3 - *Thresholds in Apdex zones*

Description

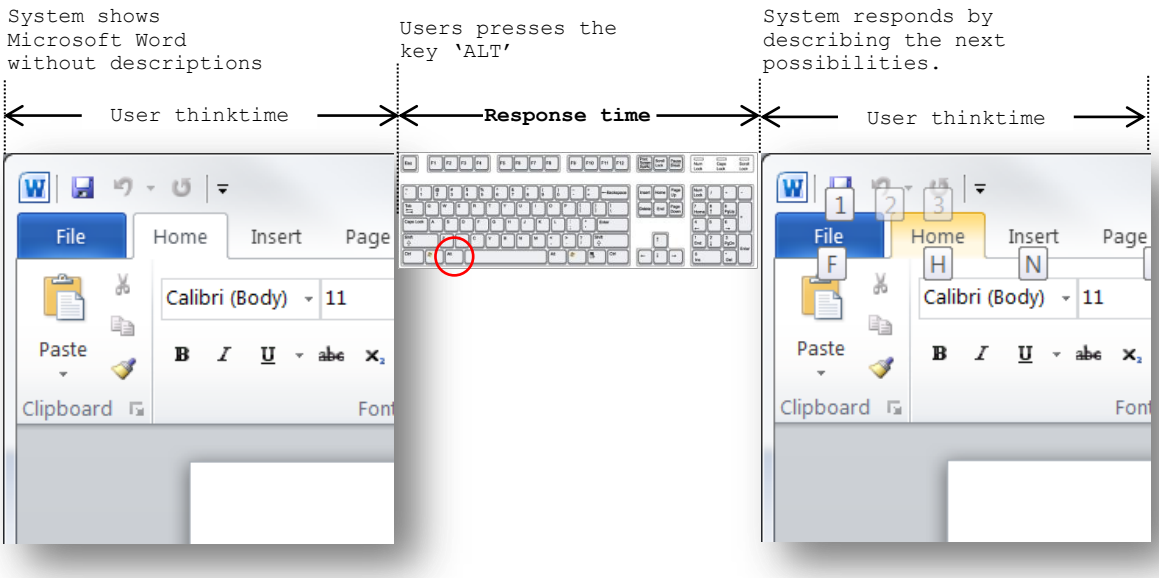
An action initiated by a user [a] which is followed by a (series of) complex action executed by the system which will (eventually) show the result [b] of that complex action. (*The user is conscious about the fact that a complex operation is being executed by a category B response, e.g. the display of a progress bar*).

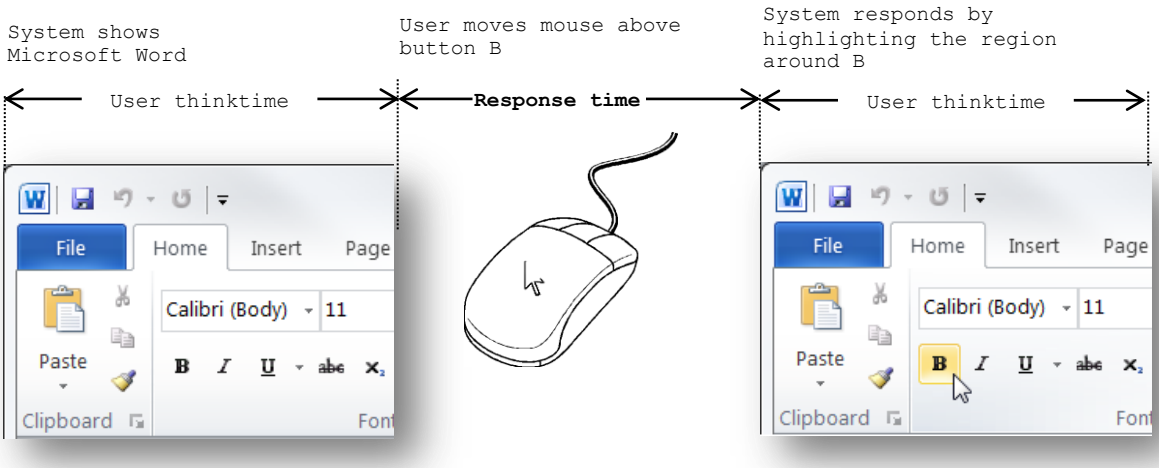


As long as the response time remains within the thresholds, the user is motivated to remain focused on the screen and wait for the result. If the threshold is exceeded the user will shift the focus to a (completely) different task.

4.3 Examples of User Actions

Category A





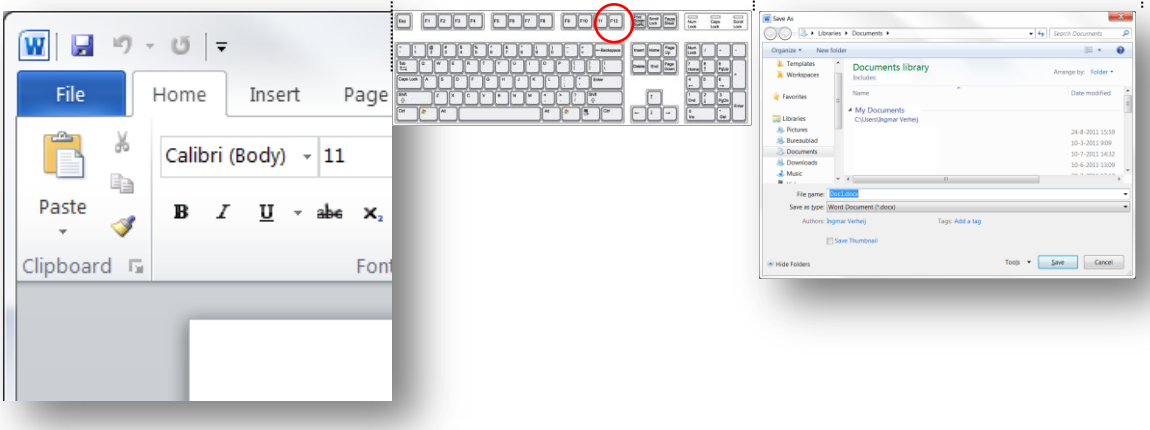
Category B

System shows Microsoft Word without descriptions

User presses the key 'F12'

System responds by displaying the dialog 'Save as'

User thinktime — **Response time** — User thinktime

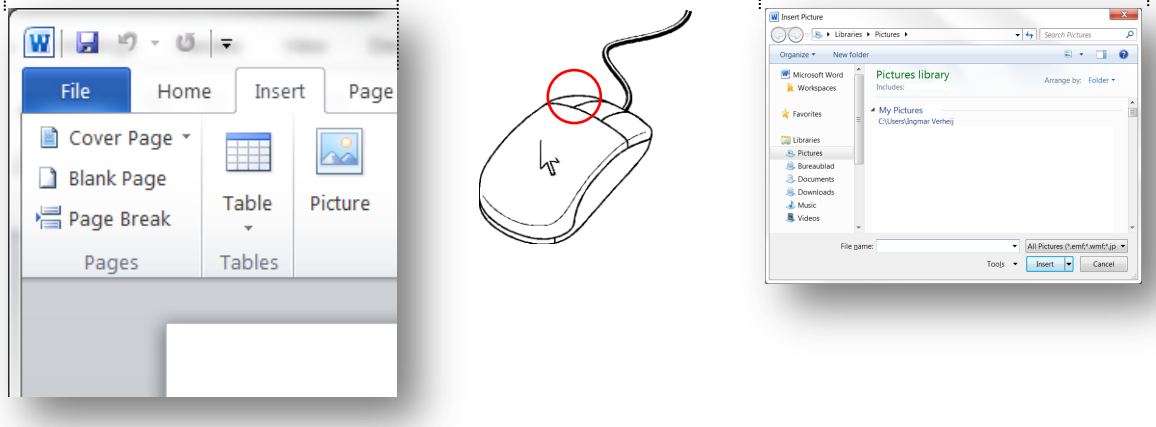


System shows Microsoft Word in menu 'Insert

User clicks with mousecursor on 'Picture'

System reponse by displaying the dialog 'Insert picture'

User thinktime — **Response time** — User thinktime



4.4 Learning Effect

Computer systems are usually built to optimize the delivery of a result. By optimizing processes a decrease in resource consumption is achieved which results in improved performance and productivity. Caching mechanisms are one of the optimization techniques applied in modern systems. Optimization techniques, like caching, are able to improve performance by remembering the behavior of a user. If the system can predict what action will be used, or has been used recently, it can prepare the action by storing it in a faster memory space. In other words, a learning effect occurs when tasks are predictable which in turn helps increase system performance. There is very little software implementing "predictive" behavior, and software that does usually uses a caching mechanism.

The learning effect can be seen if a certain action sequence is repeated multiple times. A test is conducted with a basic application that tests the response time of a system with both category A and B actions.

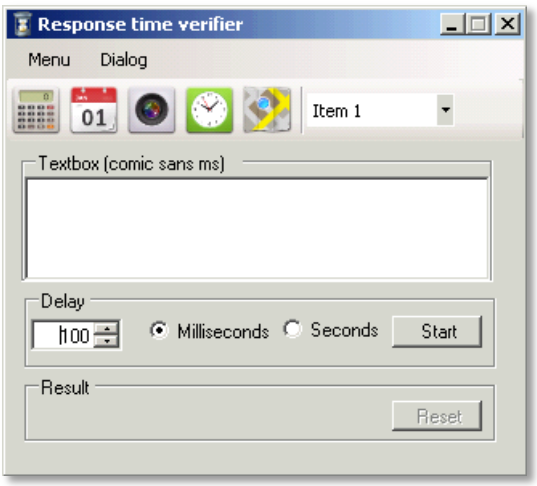


Figure 6 - Response time verifier

Based on the outcome of the test conducted the turning point lies around three iterations. In most cases the time required to perform the same action is reduced in the first three sequences. After three iterations the time required to perform the same action remains approximately the same.

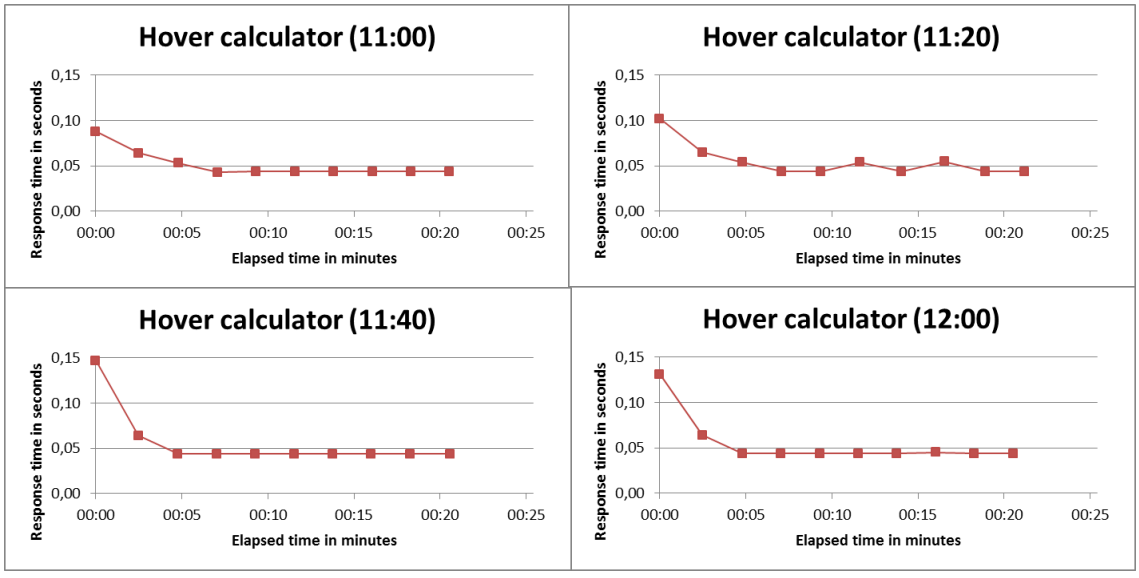


Figure 7 - Response time of category A action

Settling time and pre-run
A user starts by initiating (and authenticating) a session after which the system logs on the user. During this phase a number of processes are active in the background which might cause a degration of performance. The influence of these processes can be prevented using a settling time. The settling time allows the system to "settle down" after a user login.

The performance gain (as result of the optimization techniques) remains present during the session or until buffers are re-used for other processes. When one test is run prior to the measured test (a pre-run) the time required to perform the same actions remains approximately the same.

A test is conducted with a settling time of 5 minutes allowing background processes to "settle down". The test is then repeated multiple times without ending the session.
The results show that the learning effect occurs after the 5 minutes settling time and that the gained benefits remain present during the session.

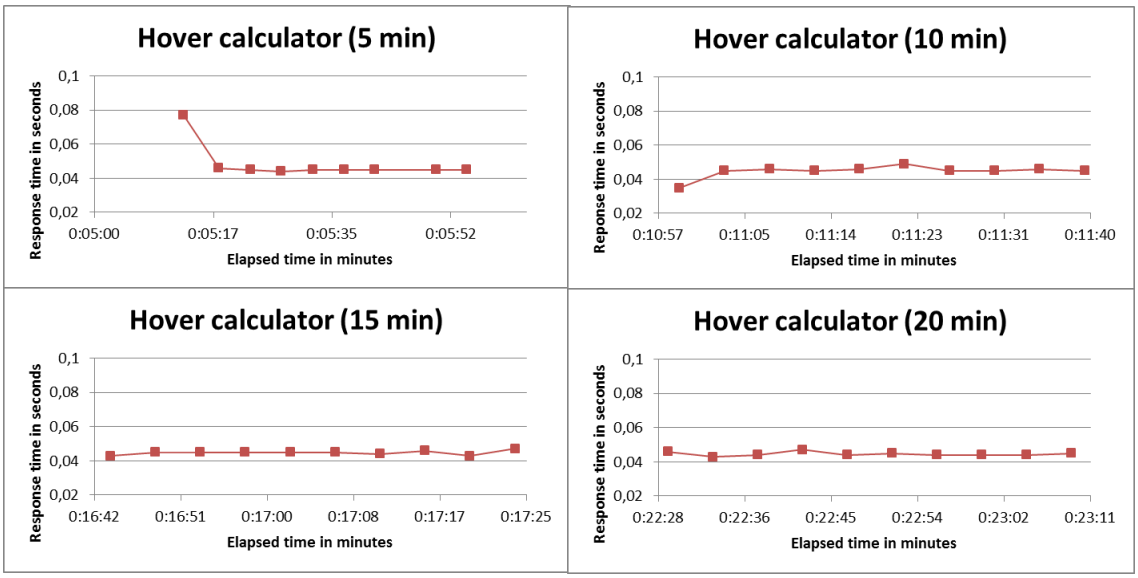


Figure 8 - Response time of a Category A action

5 Process

The collected response times, and their matching thresholds, should be calculated into a result which can be used to rate the (virtual) application. In order to compute the result, a series of response times is required. The more response time measurements, the more reliable the results are considered by statisticians. More response times can be collected by repeating the same task over and over again. More repetitions, or iterations, may lead to a "better" average but has a significant downside.

The more repetitions are executed, the more a learning effect occurs, resulting in an optimized result. Although these optimization techniques have benefits for a user, repeating the same action over and over again (in a short time period) is not realistic since this is not how users work. To determine the actual response time of an application, a high number of iterations should be avoided. Besides preventing the influence of (unrealistic) optimization benefit, it is beneficial to run a short test rather than a long test to get a quick insight in the responsiveness of an application.

The process requires generating two performance results; for application response and perceived performance. Both of the results are calculated using the Ardex formula. In this chapter we discuss how these results are calculated.

5.1 ARI – Application Response Index

The ARI rates the responsiveness of an application using response times. For category A and B, as described in chapter 4.2, the associated user actions are computed *per user action*. In other words, for each user action a result is calculated. The highest result (in seconds) is the result of the category.
The result is computed using the Ardex formula (chapter 5.3) and results in a relative number between 1 and 0. The category with the lowest score (between 1 and 0) is the overall score of the application.

In order to get a good impression of the responsiveness of an application at least 10 user actions are required per category. The user actions are executed in sequence and then repeated. This execution sequence would look like this:

> *Action 1 → Action 2 → Action 3 → Action 4 → Action 5 → Action 6 → Action 7 → Action 8 → Action 9 → Action 10 → Action 1 → Action 2 → Action 3 → etc.*

For the ARI the response time of *first three iterations* is used. By using the result of the first three iterations the learning effect is *minimized* and yet an average response time is possible. From the three response time the *mean average* is used to get a weighed result.

The result is displayed using the Ardex formula, which returns a relative number between 1 and 0, and rated using the Apdex rating tower. The formula and rating tower is explained in chapter 5.3.

Summary

Calculate the mean average response time (for the first three iterations) per user action;
↓
The user action with the highest response time is the ARI for the category (A or B);
↓
Compute the relative score using the Ardex formula and category thresholds (T and F as described in chapter 4.2);
↓
The lowest (or worst) score of category (A or B) is the overall ARI of the application.

Example
The example shows 3 (of at least 10) user actions per category to get a quick glance of the ARI calculation. The response time of the first three iterations is shown, for each user action the mean average is calculated.

Category A	Iteration 1	Iteration 2	Iteration 3	Mean average
Hover Calculator	0.125	0.110	0.100	0.112 $\frac{0.125+0.110+0.100}{3}$
Hover Calendar	0.120	0.109	0.101	0.110 $\frac{0.120+0.109+0.101}{3}$
Hover Camera	0.127	0.112	0.102	**0.114** $\frac{0.127+0.112+0.102}{3}$

Category B	Iteration 1	Iteration 2	Iteration 3	Mean average
Open color dialog	1.025	1.011	1.000	1.012 $\frac{1.025+1.011+1.000}{3}$
Open folder browser dialog	1.031	1.012	1.011	**1.018** $\frac{1.031+1.012+1.011}{3}$
Open font dialog	1.020	1.009	0.992	1.007 $\frac{1.020+1.009+0.992}{3}$

The user action with the highest average response time is used to calculate the ARI for the category. The user action with the highest average response time for category A is **0.114 seconds** (Hover Camera), and **1.018 seconds** for category B (Open folder browser dialog).

	Response time	T	F	ARI (Ardex formula)
Category A	0.114	0.100	0.400	**0.98**
Category B	1.018	1.000	4.000	1.00

Using the Ardex formula the score, a number between 1 and 0, is calculated. The score returned by the Ardex formula (chapter 5.3) depends on the parameter T and F as described in chapter 4.2.

A = 0.100 **B** = 0.400
C = *2B-A = (2 x 0.400) – 0.100 = 0.700* **X** = 0.114

$$\textbf{Ardex} = \frac{\textbf{C} - \textbf{X}}{2(\textbf{B} - \textbf{A})} = \frac{0.700 - 0.114}{2 \times (0.400 - 0.100)} = \frac{0.586}{2 \times 0.300} = \frac{0.586}{0.600} = \textbf{0.98}$$

The category with the lowest score is the overall result of the application. In this case the overall score is **0.98** (category A) which equals "**Excellent**".

5.2 PPI – Perceived Performance Index

The PPI rates the performance of an application as it would be perceived by users. The PPI varies from the ARI in that it incorporates the variability of the response times measured. The variability might affect the perceived performance by a user over a longer period of time.

Variability occurs when delays appear at random, for instance due to congestion on a WAN connection or when resources have to compete (CPU, memory, disk, etc.). When these delays increase in size and turn into big spikes, the user might perceive a long delay rating the system as slow.

The chart on the right side shows two series of data. Both series have an average (both mean as median) of 100 milliseconds, but the spreading is different.

The response times in the first series almost all (80%) are 100 milliseconds and there is not much variation (+- 20 ms). In the second series the spreading is more than +-50ms leading to more variability.

The data captured in series 2 is more likely to have experienced a random delay caused by resource congestion.

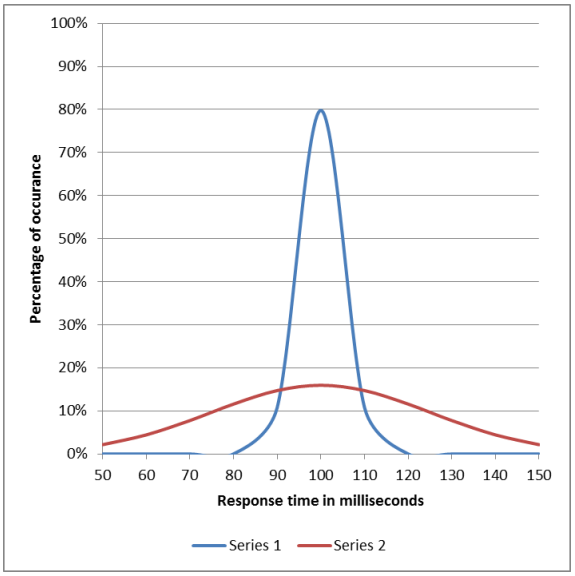


Figure 9 - Response time of two series

Since these delays occur at random over a longer period the PPI is measured over 10 iterations.
When the responsiveness is tested with the "Response time verifier" 10 iterations are completed, which results in a total time of roughly 1 minute. This is ideal for testing the variability.

The PPI is calculated by adding the Mean Absolute Deviation (MAD) [8] of a user action to the associated response time calculated for the ARI. This means that the PPI is never higher than the ARI. The subset for the PPI is the *10 response times* measured *per user action*.

The result is displayed using the Ardex formula, which returns a relative number between 1 and 0, and rated using the Apdex rating tower. The formula and rating tower is explained in a chapter 5.3.

Summary

Calculate the MAD (over 10 iterations) per user action and add the associated response time calculated for the ARI;

↓

The user action with the result is the PPI for the category (A or B);

↓

Compute the relative score using the Ardex formula and category thresholds (T and F as described in chapter 4.2);

↓

The lowest (or worst) score of category (A or B) is the overall PPI of the application.

Example

The example shows only category A to get a quick glance of the PPI calculation. For 3 (of at least 10) user actions the response time of 10 iterations is showed. For each user action the mean average and the MAD is calculated.

Cat A	1	2	3	4	5	6	7	8	9	10	Mean	MAD
Hover Calculator	0.125 *+21*	0.110 *+6*	0.100 *-6*	0.102 *-2*	0.099 *-5*	0.101 *-3*	0.101 *-3*	0.100 *-4*	0.101 *-3*	0.098 *-6*	0.104	0.006
Hover Calendar	0.120 *+17*	0.109 *+6*	0.101 *-2*	0.098 *-5*	0.099 *-4*	0.100 *-3*	0.101 *-2*	0.097 *-6*	0.098 *-5*	0.102 *-2*	0.103	0.005
Hover Camera	0.127 *+18*	0.112 *+3*	0.102 *-7*	0.101 *-8*	0.125 *+16*	0.120 *+11*	0.100 *-9*	0.099 *-10*	0.101 *-8*	0.100 *-9*	0.109	**0.010**

The MAD (see chapter 3.4) is added to the response time calculated for the ARI in the previous step (see ARI example).

Category A	Response time (ARI)	MAD	Result
Hover Calculator	0.112	0.006	0.118
Hover Calendar	0.110	0.005	0.115
Hover Camera	0.114	0.010	**0.124**

The user action with the highest result (ARI response time + MAD) is 0.124 seconds (Hover Camera).

	Result	T	F	PPI (Ardex formula)
Category A	0.124	0.100	0.400	**0.96**

A = 0.100 **B** = 0.400
C = *2B-A = (2 x 0.400) – 0.100 = 0.700* **X** = 0.124

$$\textbf{Ardex} = \frac{\textbf{C} - \textbf{X}}{2(\textbf{B} - \textbf{A})} = \frac{0.700 - 0.124}{2 \times (0.400 - 0.100)} = \frac{0.576}{2 \times 0.300} = \frac{0.576}{0.600} = \textbf{0.96}$$

The overall score is calculated using the Ardex formula; in this example the score is **0.96** which equals "**Excellent**".

5.3 Ardex

Although the Apdex method (as described in chapter 3.3) seems perfect to rate the responsiveness of an application it has one major drawback: All measurements are first placed in one of the three zones and then the score is determined without considering the value of the results. This leads to a non-granular result. Since most response times will be placed in the 'Tolerating' zone (the response time is greater than T) the Apdex formula would result in a score of 0.50, which equals 'Poor'.

To accommodate granularity a new formula is defined using Apdex as a starting point: Ardex. The Application Responsiveness Index (or **Ardex**) formula results in a 1 (Excellent) for all values <= T and, a 0 (Unacceptable) for values => F. When the response time is between T and F a linear result is returned with between 1 and 0 a maximum of 2 decimals.

The Ardex formula is first defined in this paper.

Ardex formula

$$0 < X < A = 1$$

$$A < X < C = \frac{C - X}{2(B-A)}$$

$$X > C = 0$$

Where

A = Threshold T
B = Threshold F
C = 2B – A
X = Response time to rate

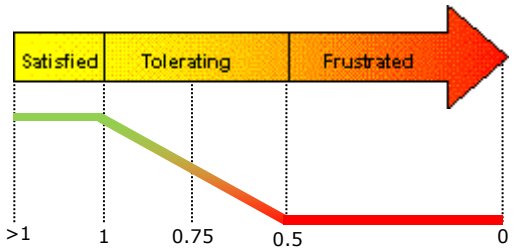


Figure 10 - Ardex results in Apdex zones

5.4 Summary

So the method returns two results; the ARI and the PPI. ARI rates the responsiveness of an application at a given time; PPI includes the variability of the response times over a longer period to determine the spreading of the results.

Both results are calculated using the Ardex formula which require an upper and a lower threshold, respectively T and F in the associated category.

User actions are categorized in either category A or B, category C is not used to determine the responsiveness of an application. The upper threshold T is the time where users start noticing a delay and the lower threshold F is the moment where users start to get frustrated, which is unacceptable.

6 Examples

The result of the process (ARI and PPI) can be used in multiple scenarios with a different approach. In this chapter I will show a few examples of how the process can be used to compare and/or rate an application and how the results can be used.

6.1 Physical desktop – SBC - VDI

Migrating from a physical desktop to a virtualized desktop, whether that's a hosted shared desktop (Server Based Computing) or a hosted virtual desktop (VDI), comes with a cost. By moving the execution of processes from the local physical device to a remote virtualized device the in- and output needs to be transported.

On a local physical device this is done easily because the operating system can communicate directly with the hardware (human interface device or graphical device). But when the desktop is moved to a remote location a remoting protocol is required. No matter how good the remoting protocol is, the chain from start to finish is bigger and more complex.

When delivering a remote desktop the user experience is a key deliverable. By comparing the performance of an application hosted on a physical desktop compared to a virtualized desktop, the impact of the change can be quantified. Since the responsiveness can be rated (Excellent, Good, Poor, etc.) the decision to migrate to virtualized desktop may be better substantiated one.

6.2 Native Installed Application - Virtualized Application

Applications running on a desktop are traditionally installed on the operating system. To overcome issues with compatibility, delivery, security or accountability applications are virtualized. By virtualizing an application an additional layer is added, which might affect the performance of an application.

To determine the impact of virtualizing an application the performance of an application, as perceived by an average user, should be determined. If performance degradation is measured the impact of the degradation can be rated. An increase in response time, caused by virtualizing an application, might change the rating from Excellent to Good. By rating the responsiveness a substantiated decision can be made to virtualize an application.

6.3 Datacenter - Remote Site

Virtual desktops are often hosted in a (local) datacenter where users are spread across multiple remote locations. The remote locations are connected via a MAN or WAN connection to the datacenter providing access to the desktop.

The impact of bandwidth or congestion on a connection can be significant on the remoting protocol delivering the desktop to the user. Although the performance of the systems hosting the desktops / applications in the datacenter may be adequate, the users in a remote site might encounter a degraded performance.

By determining the perceived performance (PPI) of users in a remote site, including the effects of all items in the chain (including latency and jitter), a better understanding is achieved.

6.4 Turning Point when Scaling

Scaling (whether it concerns a virtual desktop solution, fileserver, mail server, etc.) is important to validate a design and the capacity of a system. Scaling is done by conducting a stress test resulting in a turning point where resources have to compete and the performance degrades.

The more resources are used, the more efficient the system is (are unused resources a waste or good practice?). But competing resources eventually lead to congestion resulting in a loss of performance. Finding the point where the performance is degraded to an unacceptable level is essential to scale properly.

Scaling can be done by just looking at the computational performance, for instance when cpu usage is over 80% or the disk queue length is too high, but also looking at the perceived performance. Computational performance and perceived performance are related, but there is no guarantee that if the performance metrics remain within their thresholds (defined by technician) the perceived performance is acceptable (defined by decision maker).

6.5 Service Level Agreement

A Service Level Agreement (SLA) is an agreement between a provider and a customer to make sure the customer receives the service it pays for. An SLA consists of multiple key performance indicators (KPI) that monitor different aspects of the delivered service. When a threshold is exceeded the SLA provides in a compensation of some sort.

The perceived performance of an application (delivered via a remote desktop) can be a KPI in an SLA. By monitoring the perceived performance the quality of the service can be determined. Because the ARI and PPI results are rated based on an average user, an agreement can be made about the quality of the delivered service that is better aligned to the productivity of users, rather than only considering traditional computational metrics because – as applications increasingly rely on multiple platforms – delivered as cloud/cloud like services, knowing the resource usage of an individual component is less important than the service's ability to deliver data and resource in a timely manner.

7 About

7.1 The Author

Ingmar Verheij is a Senior Consultant at PepperByte primarily focusing on virtual desktop (SBC/VDI), system and workspace management and (performance) monitoring.

Ingmar was born in 1980 and studied Computer Sciences in Utrecht (B ICT). He started his career as a Systems Engineer at Actacom. After years of gaining experiences Ingmar joined PepperByte as a consultant in 2008.

To contact Ingmar directly, send an email to i.verheij@pepperbyte.com, or follow Ingmar on twitter: @IngmarVerheij.

7.2 PepperByte

PepperByte specializes in the fields of Server Based Computing, System Management, Virtualization, Storage and Networking.

PepperByte supports clients with advice, development, implementation and management of their IT systems. PepperByte is independent; we do not supply hardware or software. We therefore offer independent and tailored advice to our clients.

Each PepperByte specialist can call on a range of knowledge and experience in order to successfully and profitably fulfill your assignment. All our specialists are assisted and supported in attending necessary courses, and obtaining accreditation within the core fields, for instance for Microsoft, Citrix, Cisco, RES Software, VMware, ITIL and Prince 2.

The experience and certification of our specialists makes PepperByte a strategic partner of major vendors including Microsoft, Citrix, RES, VMware and DeNamiK.

7.3 Special thanks

Special thanks go out to the guys who were willing to review this paper before publishing. Although many hours are spend researching and designing the process, and finally writing the paper, an objective review is valuable. Especially when a review is done by people who have experience in the same industry and extensive knowledge about the subject. I have great respect for all the reviewers and would like to thank them all for their contribution to this paper.

Baggerman, Kees	Technical Consultant at Inter Access.
Garling, Okke	Performance test consultant at Perfcon B.V.
Hokke, Marco	Developer at PepperByte;
Mangan, Tim	President at Virtualization Boston; Kahuna at TMurgent Technologies LLP.
Nikolic, Daniel	CTO at DeNamiK; CTO and Infrastructure Architect at PepperByte.
Tielen, Jeroen	Senior SBC/VDI Consultant at PepperByte.
Wood, Andrew	Director, Gilwood CS Ltd; Citrix Project Consultant at tuCloud.

8 References

1. **Tim Mangan**, "*Perceived Performance – Tuning a system for what really matters"*, TMurgent Technologies. September 18, 2003. http://www.tmurgent.com/WhitePapers/PerceivedPerformance.pdf

2. **Tim Mangan**, "*Perceived Performance Reloaded – Tuning a system for what really matters"*, TMurgent Technologies. May 10, 2011. http://www.tmurgent.com/WhitePapers/Perceived_Performance_Reloaded.pdf

3. **Robert B. Miller**, "*Respons time in man-computer converstational transactions"*. International Business Machines Corporation. December 1968. http://portal.acm.org/citation.cfm?id=1476628

4. **Ben Schneiderman**, "*Response Time and Display Rate in Human Performance with Computers"*. University of Maryland. September 1984. http://portal.acm.org/citation.cfm?id=2517

5. **Jakob Nielsen**, "*Usability Engineering"*. Morgan Kaufman, 1993. http://www.useit.com/jakob/useengbook.html

6. **James R. Dabrowski, Ethan V. Munson**. "*Is 100 Milliseconds Too Fast"*. CHI 2001 http://portal.acm.org/citation.cfm?id=634255

7. **Peter Sevcik**. *"Defining The Application Performance Index"*. Business Communications Review, March 2005. http://www.apdex.org/docs/Defining_The_Application_Performance_Index.pdf

8. **Wikipedia**. *"Absolute deviation"*, Wikipedia, consulted on july 18. 2011, http://en.wikipedia.org/wiki/Mean_absolute_deviation

9. **Apdex Alliance Inc**. *"Apdex Technical Specification".* Apdex Alliance, Inc. January 22 2007. http://www.apdex.org/docs/Defining_The_Application_Performance_Index.pdf