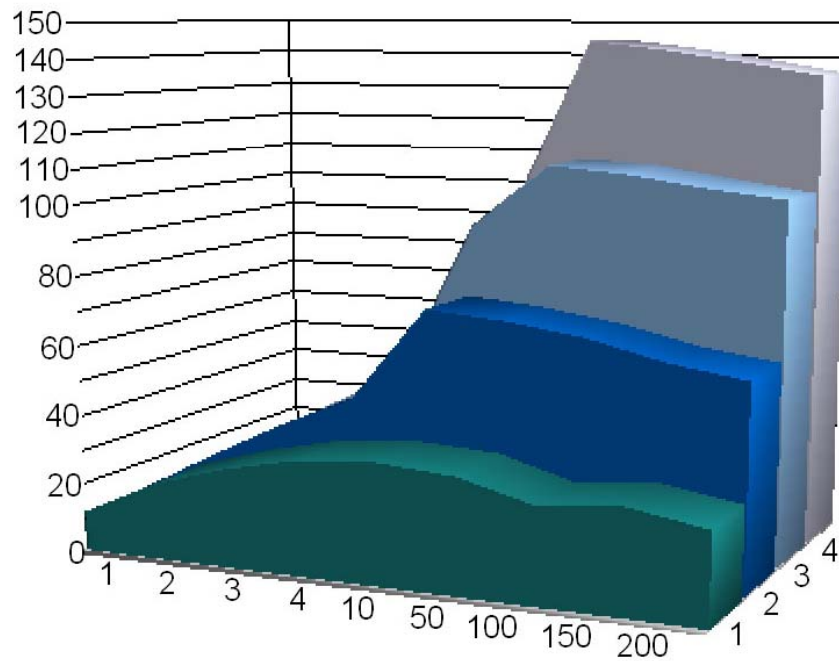


Add *optimal*



to your performance testing



**TEST 2008  
INDIA**

**Authors**

***Abhishek Talwar***([abhishek@adobe.com](mailto:abhishek@adobe.com))

**&**

***Abhinav Agarwal***([abhinava@adobe.com](mailto:abhinava@adobe.com))

**Adobe Systems India Pvt. Ltd.**

**Noida**

# Table of Contents

<b>Topics</b>	<b>Page Number</b>
What is Performance Testing	3
Performance Testing objectives	3
Automation of performance testing	3
How to improve your product performance	3
Performance test planning	4
Implementation of the above test plan	7
Performance automation case study	10
Performance dos and don'ts	11
Conclusion	12
Bibliography	12
Biography	13

## **What is Performance Testing?**

As per Wikipedia; **performance testing** is testing that is performed, from one perspective, to determine how fast some aspect of a system performs under a particular workload. It can also serve to validate and verify other quality attributes of the system, such as scalability, reliability and resource usage. To conduct performance testing is to engage in a carefully controlled process of measurement and analysis.

## **Performance Testing Objectives**

Performance testing can be viewed as a “black box” testing, which focuses on application and system behavior from outside with no knowledge of the program code that supports the system. The goal of performance testing is not to find bugs, but to eliminate bottlenecks and establish a baseline for future regression testing. The main objectives of the performance testing are:

- 1) The maximum number of concurrent users that can be supported while offering “Acceptable performance”.
- 2) The maximum number of concurrent users that can be supported prior to causing a system failure.
- 3) The location of bottlenecks within the application architecture.
- 4) The impact of a software or hardware change on the overall performance of the application.
- 5) To benchmark applications for given combinations of software and hardware.

## **Automation of performance testing**

Automation of performance testing has been there for long. There are many tools in the market that can be used to measure performance of a product or features. The discussion of these is out of scope for this paper, however, we want to touch base some of the performance automation best practices followed at Adobe which is helping us in performance testing.

Test automation of performance is done for many reasons.

- It can save time
- It gives very accurate results
- Some parameters have performance values which are not easily detectable by humans, in such cases introduction of automation into performance testing is a necessity.
- Comparison and analysis of results is not prone to human errors
- Make testing easier
- Improve the testing coverage, so that performance test results are better depiction of the actual picture.
- As for any automation type... It can also help keep testers motivated

## How to improve your product performance?

### Define high level goals!!

A sample of high level goals to be set both at code level as well as workflow level in order to make your product performance better:

#### 1) *Reduce disk access*

- Read in large contiguous blocks and avoid seeks.
- Directory look up, file access within a file should be minimized.

#### 2) *Code, data reads* On-demand loading

- If on-demand mechanism is not feasible, delay reads/execution until later

#### 3) *Set clear goal*

- Clear definition of goal should be set (e.g. 1 second from launch to displaying the first page on machine XYZ)
- Use reproducible tests, have standard set of testing and test docs.
- If you do everything, your cold time should be close to warm time

#### 4) *Measurement*

- Create baseline for accurate measurement.
- Automate everything. Build in error (time increase) detection as well as any result which may help investigation

#### 5) *Management commitment* (This may be the most important thing on this list.)

- Allocate resource on setting up, running the automated tests
- Allocate resource for monitoring daily tests and investigating issues
- Commitment to address performance issues over implementing a features.
- Make sure engineers understand a fraction of second is too long.

### Define what falls in your purview

Before the start of the project one should have a list of parameters that one needs to track and improve upon. This list should be made considering two factors:

- a) Factors which affect product performance the most (efficiency parameter/ product Bottlenecks)
- b) Factors which hurt the user the most (user perspective/workflow)

Some of the factors that once can take as a starting point are:

*BaseAddressCheck, ComponentDiskBreakdown, DiskHitType, DllCheck, DllsByLabel, flippages1\_win\_rdr, Fragmentation, IdleProcs, images-spook, LaunchCheckerMacBC, LaunchCheckerMacBW, LaunchCheckerMacC, LaunchCheckerMacW, LaunchFileIO, LaunchSpeed, LaunchSpeedFL, LaunchSpeedFLWarm, LaunchSpeedIE, LaunchSpeedIEWarm, LaunchSpeedTotal, LaunchSpeedTotalMac, LaunchSpeedWarm, munisystem, OpenFile, Panning\_Spook, Panning\_truck, Prefetch Parser, QuitSpeed, ReadLogger Launch, ReadLogger Launch FL, ReadLogger Launch oro, ReadlogWithTiming, Type3 Fonts, ViewingI, ZStringLaunchStrings.*

## **Performance test planning**

A sample test plan specific to performance testing (includes both manual as well as automation efforts)

### ***Objective***

Performance testing is critical for the success of the product. The intent of this document is to identify crucial areas for Performance measurement, and to come up with a plan which would be followed by all the teams in their areas of testing.

### ***Scope of testing (factor name - Priority)***

Performance testing for the product can be divided into following categories:

- File Launch – High
- File I / O – Very High
- UI Performance – Medium
- Resource intensive tasks performance – High

### ***Team Responsibilities***

Performance Test team would be responsible for the following:

- Testing of core functionalities – Application Launch, File I/O
- Major enhancements in Performance, such as Resource intensive tasks (focus for this product version release).
- Helping Quality Engineers in automating their performance test cases.
- Develop and maintain automation framework for reporting test results.

Almost all the features would have some form of Performance testing.

We need to identify the critical test cases which are relevant for Performance testing.

It will be the responsibility of each Quality Engineers to **identify** Performance test cases of the areas that they are testing. We can help the teams in integrating the code for measuring the time and reporting the data in a standard framework.

### ***Quality Goals***

- Inputs to be taken from Product Management for targeted improvement in performance.
  - The performance, in no way, should be less than that in the last release. The last release would be the major benchmark for performance measurement.

### ***Creating Benchmarks***

- We need to create new benchmarks as the performance results of last release would have been on a different machine (just to be sure that we compare apples with apples and not apples and oranges)
- Take the average of at least 10 readings, so that benchmark is accurate. Disregard any extreme readings to compute the average
- Ensure that no other application is in use (including Virus scan) as that would distort the readings (refer to the '*do's and don't*' bullets given later in the paper.

### ***Test Environment***

Consistent Test Environment will be used.

- Use same machine for running the Performance test.
- Hardware: 1GB RAM machine would be used for regular testing. However do test on low end (512 MB) and high end (2GB) near the milestones.

- Software: To maintain consistency, no other program should be running on the machine. The anti virus software should also be switched off (refer to the 'do's and don't' bullets given later in the paper for more details)

### ***Frequency***

- File Launch would be tested on daily builds on WIN XP and IMAC
- File I/O to be tested once a week.
- In addition, one should test on the other platforms – VISTA and MAC PPC, Low and high end configurations and Japanese platform near the milestones.

### ***Test Cases***

The standard test areas / cases would be the following:

*Application Launch*: Cold launch, Warm launch, launch after deleting preferences

*File I / O*: File New, Open, Export, Import, Save, Copy / Paste

*UI Performance* : Edit , Move , Scroll , Select / Unselect , Zoom , Pan , Preview .

### ***Test Data***

The test file(s) used for automated performance measurement should have the following characteristics:

- The test file should be representative of a real life projects.
- Should be sufficiently large for performance measurement

### ***Test Types***

In addition to the standard testing, we should also focus on the following

- Different machine Configurations, including 64 bits, Dual Processor.
- Stress testing with multiple applications running / multiple files open / Low Memory
- Complex and large files
- Opening files over the network
- Manual testing with very large files. This would, of course, be a part Of our ad hoc testing as well

### ***Automation***

Automated time measurement would be used.

*Application Launch*: Time measurement Utility would help give accurate performance numbers build over build in the order of milliseconds. The sample architecture, performance graphs and reporting mechanisms have been listed later in the paper.

### ***Goals for Performance reporting framework***

1. Store all performance results into database automatically.
2. Provide a good mechanism for displaying performance results (In form of Charts/tables) in a customizable manner.

The product Performance Automation test scripts write the results into the text files. As we need to store all the performance results into a database, a script/program will be invoked (as soon as performance test run is complete) that will import the results from text files into database.

A web application would be developed which will retrieve performance results from the database and display it to the user in form of customizable charts and tables for result analysis. Application will provide the UI where user can select various parameters (Test case, OS, build numbers etc) on which he wants to see the results in form of charts/tables. User will be able to generate 2 types of charts

1. Across Builds – Line Chart displaying time of a particular operation/test case (Launch, File I/O) across different builds on a specific platform. This chart will also contain timing of same operation on last product release as well. Since we will be taking multiple (5) readings for every operation on every build, average of 5 readings would be used in this type of chart.

2. Within Builds – Line Chart displaying various readings for a specific operation on the same build. This chart will help us in analyzing performance of the operation when it is executed multiple times.

In addition results can be displayed in form of tables when user selects a complete Area to see the general Pass/Fail statistics.

### ***Defect Logging/bug guidelines***

We can follow the following guidelines:

- Bugs related to Application Launch and File I/O entered under Product Area “Performance”.
- Performance Issues found by Quality Engineers of other teams to be entered in their respective Product Area , with Failure Type “Performance Issue”.

### ***Performance testing broad level milestones***

25% of performance test cases (covering 35% use cases): by end of 10% of cycle

50% of performance test cases (covering 62% use cases): by end of 35% of cycle

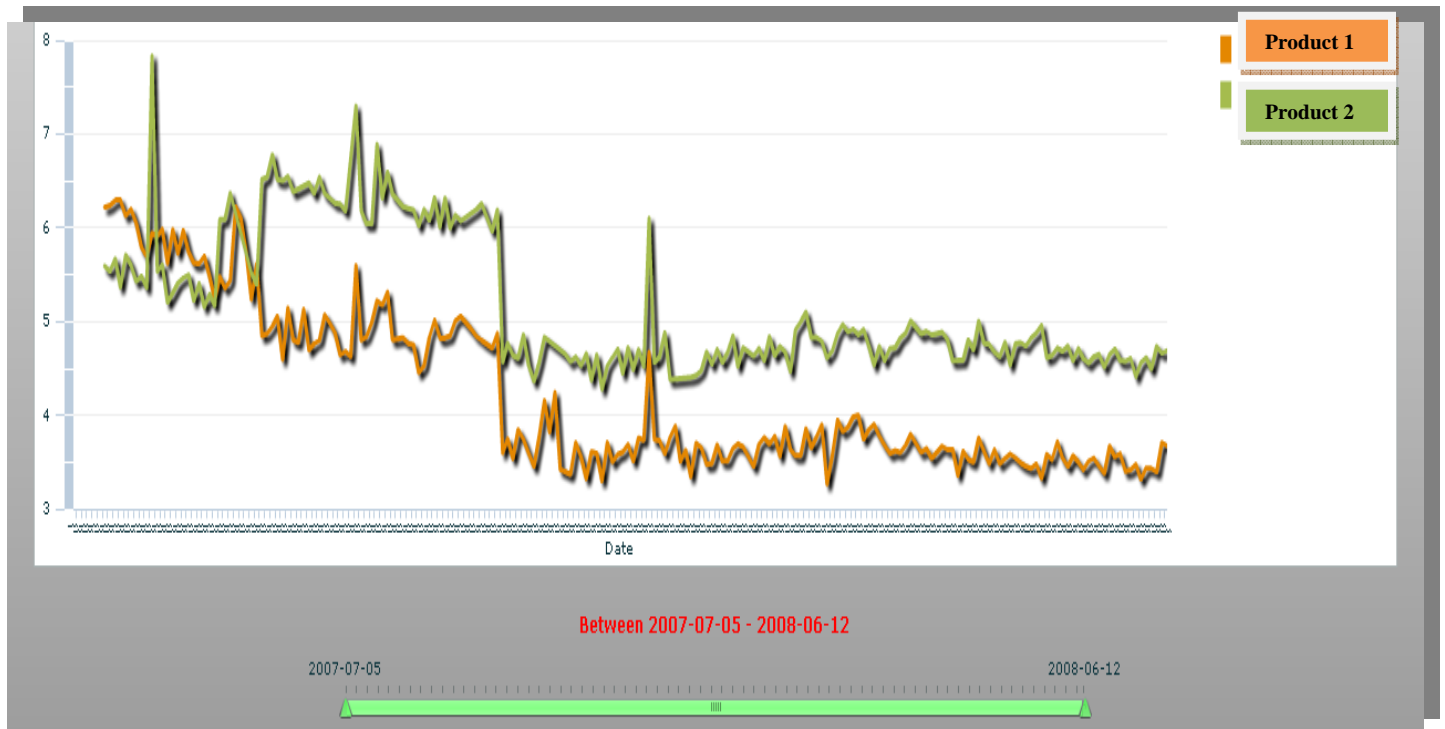
90% of performance test cases (covering 95% use cases): by end of 70% of cycle

The goal is to reach 90% coverage of performance test cases of the entire product covering more than 95% use cases.

The rest of the 30% of the cycle, there would be no development of performance test cases only baking of the product and logging defects as found by the performance testing suites.

## Implementation of the above test plan

Test Machine	Build	Hardware	OS
acroperf5	ns	2 GHz Intel Core Duo 1GB	OS X 10.4.8
acroperf6	ns	Dual 2 GHz PowerPC G5 1.5GB	OS X 10.4.8
acroperf7	ns	P4 Mobile 1.8 768MB	XP
acroperf8	ns	3.4 GHz Intel Pentium D 2046MB	Vista
acroperf9	ns	1.66 GHz Intel Core Duo 512MB	OS X 10.4.8
acroperf10	ns	1.66 GHz Intel Core Duo 2GB	OS X 10.5.1



**Figure 1: Product 1 and Product 2 behaving to the same set of performance enhancements from July 2007 to June 2008**

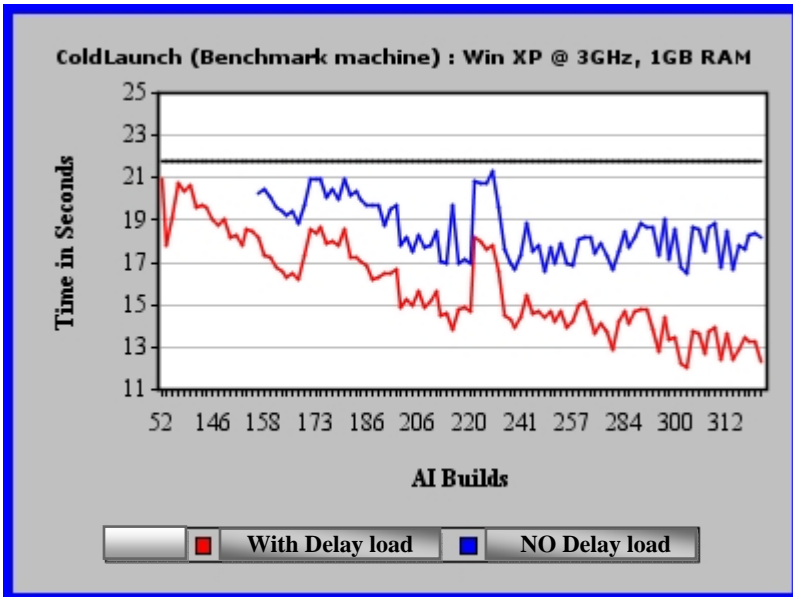


Figure 2: Variation of the same product with and without delay load

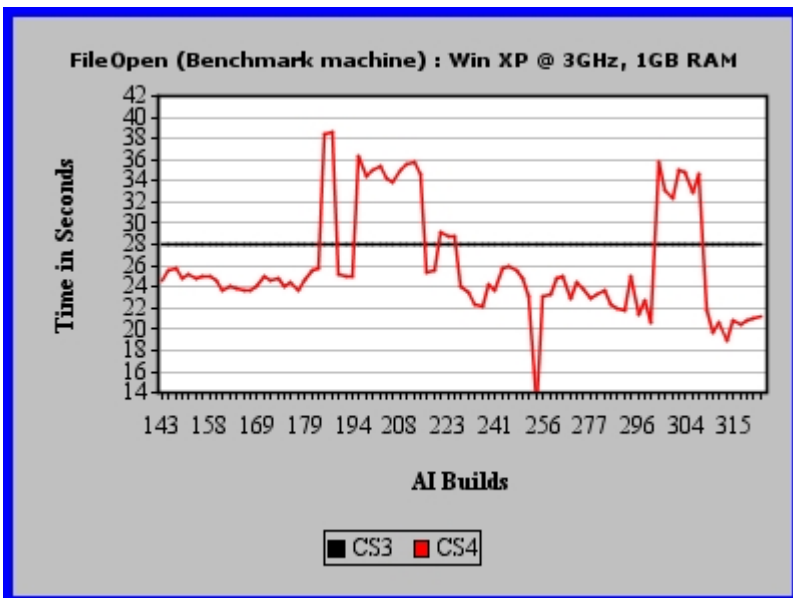


Figure 3: Variation of a product cycle at various stages of development

### Performance Charts

**Area :**

**Sub Area :**

**Test Case :**

**OS :**

**Builds :**

**Note: Please select the Performance Area, Sub-Area, Testcase and Operating System to generate charts**

Figure 4: Sample performance chart showing the various configurable parameters that user can set

#### Product Builds

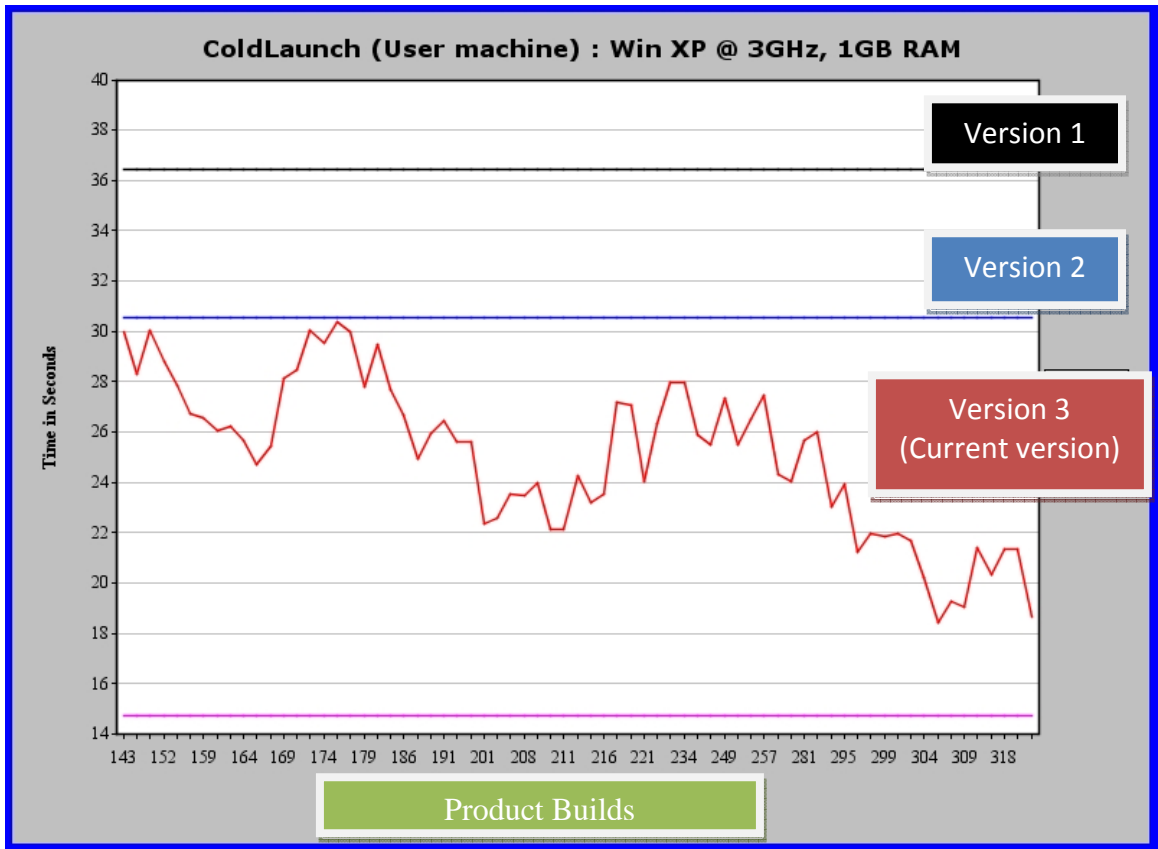


Figure 5: A graphical representation of the cold launch timings of 3 cycles of the same product. (red one being the latest release). Lower the time, better it is.

## Performance Automation case study

### Architecture details for performance automation design

The automation codes written for Performance Automation is using a 4-level architecture .They are:-

- a) Host programming language like C++/Java (as a language & its inbuilt functions)
- b) Interface functions as exposed by the Automation framework (in our case the parameters are platforms to be tested on, configurations to be picked, variables to check, pass criteria etc)
- c) Common functions (helper functions to access common functionality like database access etc)
- d) Code specific to the Product under test

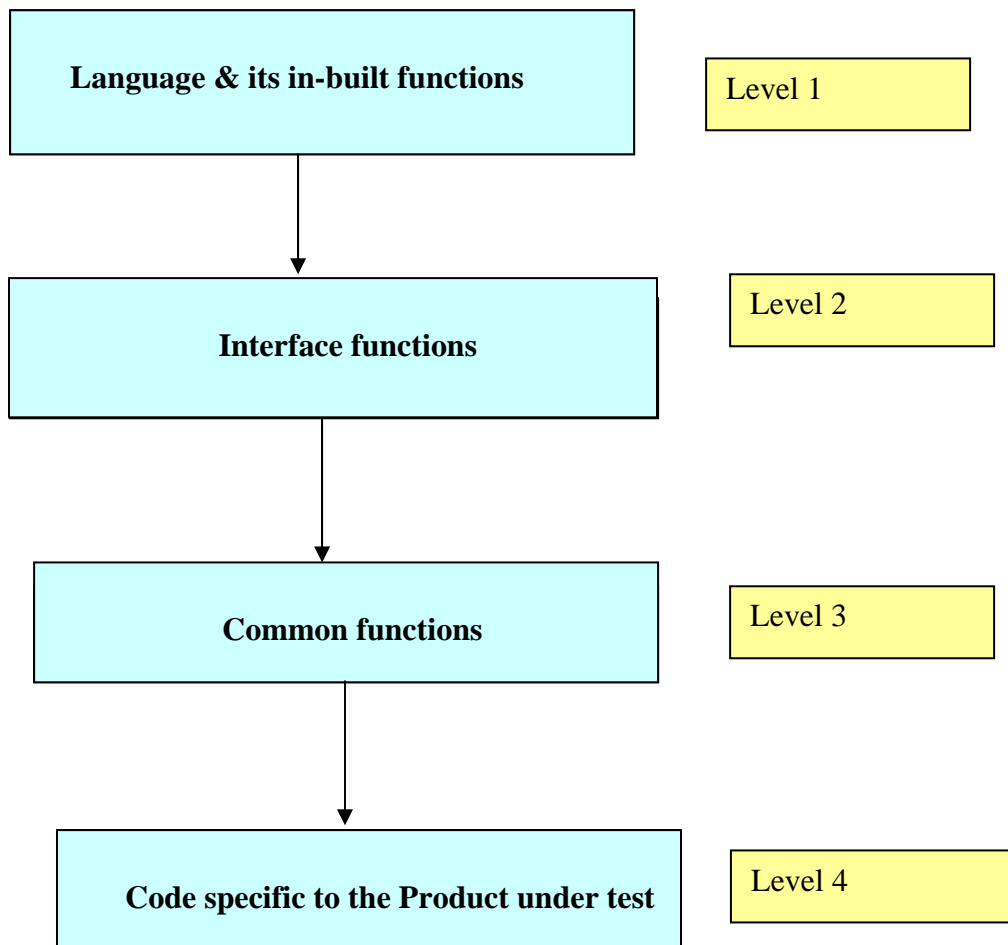


Figure 6: Diagram of how the automation code is arranged with respect to the 4 tier architecture.

At the 3<sup>rd</sup> level come some of the common functions which are used by multiple teams who are using Automation framework for automating their features. Some of the common functions that the product uses are Timer class (a common interface to measure and report time with accuracy of milli-second), Launch Product events etc.

## Sample Data

Products	Dataset 1	Dataset 2	Dataset 3	Dataset 4
Product A	107	83	534	130
Product B	151	108	568	147
Product C	281	190	0	216
Product D	133	100	589	161
Product E	0	84	544	127
Product F	245	98	0	183
Product G	89	65	356	72
Product H	111	101	450	124
Product I	0	0	0	0
Product J	0	0	0	0

Figure 7: Performance data for various competitive products on 4 data-sets

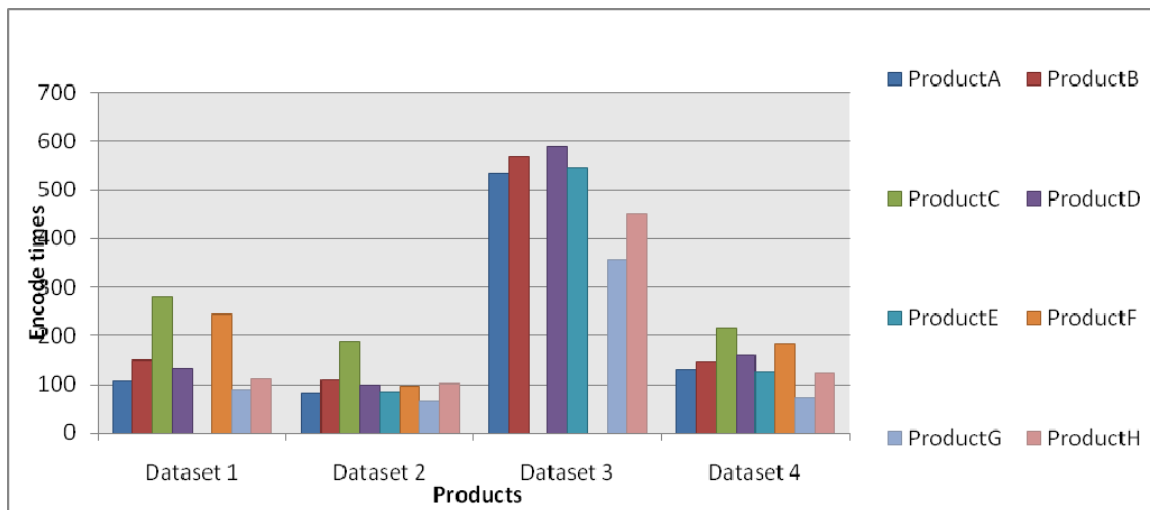


Figure 8: Graph for performance data in figure 7

## Performance Dos and Don'ts

- The most important step in performance is to ensure benchmark data is correct. In case benchmark data is not available, try to collect benchmark first
- Restart the system before doing performance testing
- Remove the network cable from the system to ensure that the system is out of the network (if this is desktop application)
- For web based application, try to use a tool () so that you use same bandwidth every run
- Remove the network cable from the system to ensure that the system is out of the network (if this is desktop application)
- For web based application, try to use a tool () so that you use same bandwidth every run
- Ensure that the disk in which temporary/ intermediate files are saved has sufficient free space

- In case these are saved at "C" drive then we should be extra cautious.
- Ensure that after installing the application whose performance is to be measured; disk has sufficient free space left (Make sure there is sufficient space available at "C" driver)
- Check that the page file settings are within limits i.e. initial size ideally should be equal to the recommended value .Also maximum size should not be too high.
- While performance testing is underway, no application should be installed/uninstalled
- Disable all other tasks (except the application whose performance is to be measured) including anti-virus software etc.
- After each reading (time taken etc.), close the application and re-launch the application for next reading (to ensure that memory consumed by the application in earlier run is now released/ free)
- Screensaver should be off
- All power settings (Turn off monitor, Turn off hard disks, System standby, System hibernates) should be set to 'Never', i.e. system should not sleep/hibernate under any case
- All test data should be copied locally so as to avoid any network issues
- In case of using the laptop disable wireless network
- Check the file size of "output" files for same benchmark; it should not vary too much

## **Conclusion**

Performance testing has become an integral part of any project that requires testing. Users today want results at lightening fast speed. The products which are best in the class owe it not just to the features that they offer but also to the performance they can fizzle out. People shifted out of hotmail to lesser known players like rediffmail and IndiatimesMail mainly because of performance issues with the Microsoft product.

Google is able to search 100+ million servers spread worldwide and return back the result in less than a second.

In-house customized automation has given a new perspective performance testing and has really pushed it to the limits.

We hope that our paper with illustrative examples from our experience can help you make a better 'performing' product from the very next release itself.

## **References**

Most of the contents of this paper are based upon the experiences of the authors in their domains with respect to performance testing.

## Biography

Abhishek Talwar is Lead Software Engineer at Adobe since last 4 ½ years. He has a total experience of around 5 ½ years in various domains like Video, Imaging and Telecommunications. He has worked on various fields of testing including White Box testing, Automation, scripting and Black Box testing. His current role involves developing tool which make the life of a tester, at Adobe, easy. Abhishek holds an international patent to his name in the field of video. He also holds a patent publication in the field of Documents. He has written five international paper publications in the field of API testing, Scripting, testing best practices, and project management. One of his submissions “API Testing: Catching hidden bugs early in cycle” was presented at the STC2005 main conference in Hyderabad. One of his papers on Quality mantras was voted as the best paper in Adobe India QE summit, 2005 and was also chosen as an article in QE quarterly newsletter.

Abhishek holds a B Tech degree in Information Technology from University School of Studies, Delhi and MBA (finance) from IMT Ghaziabad (Distance Learning). He is an ‘Indian Testing Board’ certified foundation level tester.

Email: [Abhishek@adobe.com](mailto:Abhishek@adobe.com)

Abhinav Agarwal ([abhinava@adobe.com](mailto:abhinava@adobe.com)) is Software Quality Engineer at Adobe since last 3 years. He has worked in various domains like Video, Imaging and Codec .He has worked on various fields of testing including Automation and Black Box testing. Currently he is single handedly responsible for all codec related R & D testing tasks where he is handling the entire QA alone. *For his excellent work in performance testing of his product, he was recognized by Adobe Spot Award*

Abhinav holds a B Tech degree in Electronics Technology from Harcourt Butler Technological Institute, UP along with below certifications



**CERTIFIED EXPERT**  
Premiere®

