



# Ship Your Product In Infinite Languages

Abhishek Talwar, Abhinav Agarwal  
Adobe Systems India Pvt. Ltd.  
Noida, India  
{abhishek, abhinava}@adobe.com

## Abstract:

Software of today is defined by level of customization it provides. The more customized you are, the more people love your software. Among the most important and essential customizations needed by any software user is native language support. This can be achieved by Localization. This paper would give insights on how to better localize your product to get maximum returns.

## Keywords:

Product testing, Product suites, Workflow.

## 1 Introduction

Software of today is defined by level of customization it provides. The more customized you are, the more people love your software. Among the most important and essential customizations needed by any software user is native language support. Imagine a non-English user logs into a website and finds content in English, which he doesn't understand. However, giving support to each and every language of the world isn't practically possible. Also, each language has its own set of complexities, its own unique character sets, its own problematic characters, its own byte code structure.

With each passing day, competition is increasing be it in IT sector, Telecommunication, automobile field or any other sector. More and more software companies are becoming global and so are their products. In today's global economy, a profitable product or service needs to be compatible across different markets. Operating in a single market is no longer a logical option. Take for example; Adobe's products are shipped in multiple countries. This means that these products are being used by users across the world with different cultures, different native languages, different locales and so on and so forth. So what's the big deal? Just imagine a person is in an unfamiliar country where he doesn't understand the local language. He wants to place an order in a restaurant but cannot read the menu or he wants to buy a commodity but doesn't understand the local currency. Same is the case with software that is to be made world-ready. Creating a world-ready application is a desired goal. And this is where localization of a product comes into picture. Localization has become an important aspect with the advent of globalization. Localization, sometimes shortened as L10N, is the process of adapting a

**PureConferences**  
www.pureconferences.com

Pure Conference Solution Pvt Ltd., A-108B, Sector 58, NOIDA 201301, India; Tel: +91 120 4621080

Page 1 of 11

product, application, Web etc., to a specific language by taking care of cultural, legal, linguistic and technical requirements of a specific locale.. As differences in how cultures/locales conduct business are heavily shaped by governmental and regulatory requirements. Therefore, localization of business logic can be a massive task. For example a world ready product has to cater to double-byte languages in Asia, single-byte languages in Europe and bi-directional languages in Middle East. While quality engineer in all its varying facets must be globalized if you want to ensure world-readiness, legacy testing processes will require the most adjustments to achieve this very attainable goal. The nature of these changes basically involves adding another layer to the test process that was established without globalization in mind-namely, requiring that all design documents, practices, and procedures be world-ready. It is not always an easy task, as implications and requirements of going global might not be obvious in the development process, but it is essential given the limitations of legacy testing processes. So the major goal is to ensure that a world-ready product is delivered on schedule without a compromise in the quality. But does all this mean that a quality engineer must be proficient in all languages in which application will be tested? Well we refrain from giving an answer now. In this paper we will discuss (based on our experiences while working on some of the Adobe’s world ready products ), software internationalization process & its three key issues which are globalization, localizability, and localization , testing process ,skill set required for quality engineer , challenges in testing and finally some examples. At the end of the paper our audience hopefully will be able to get the answer for the question which we left unanswered. The standout point of our paper would be to guide the reader by means of a step by step process on how to implement a strategy to be able to test a languages knowing only very few things about the language, and that too many times faster than the methods adopted today.

## 2 Internationalization

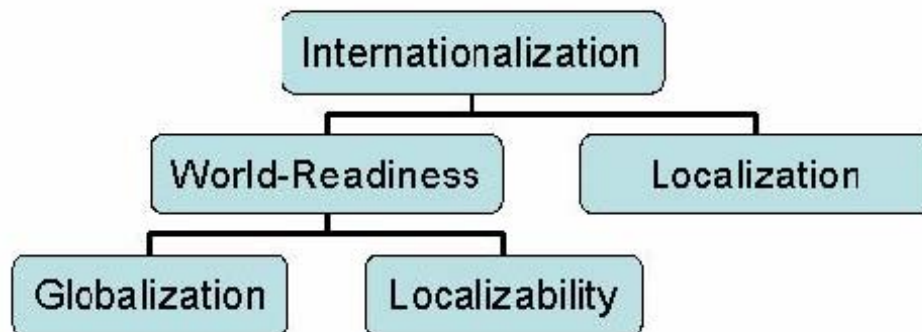


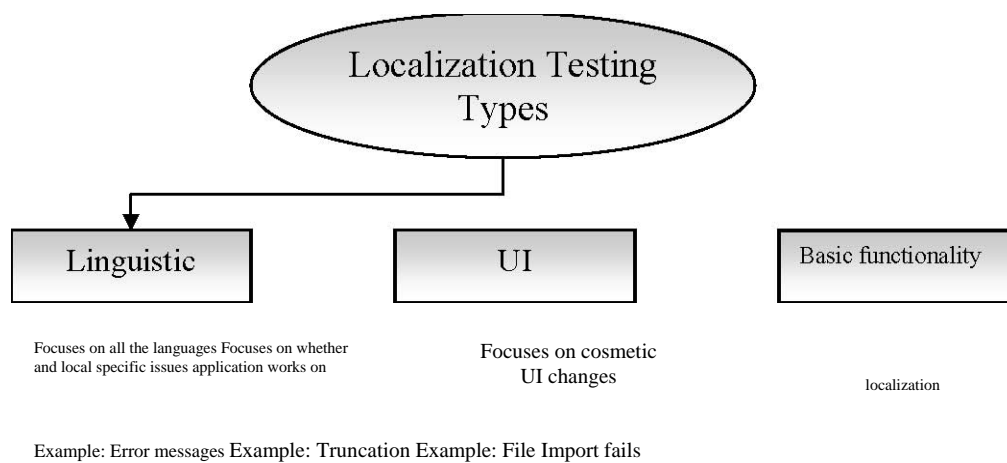
Fig 1: Broad diagrammatic view of the internationalization biosphere (courtesy Microsoft: [http://www.microsoft.com/globaldev/getwr/steps/wrg\\_g11n.mspx](http://www.microsoft.com/globaldev/getwr/steps/wrg_g11n.mspx) )

### Internationalization

Designing, building and/or re-engineering products, including hardware or software, so that, they can be adapted to regional norms without requiring subsequent engineering changes to the core application.

### Localization

It is the process of customizing the software product for each language that is to be supported. It is the aspect of development and testing relating to the translation of the software and its presentation to the end user. It includes translating the program, choosing the appropriate icons and graphics and other cultural considerations. It also may include the translation of help files and documentation. Localization is abbreviated as L10N, which means that 'L' and 'N' are separated by 10 characters.



Outsourced In-house In-house

Fig 2: Definitions/Testing Types

**Globalization** Once your product is internationalized and localized and is ready to be shipped in different markets (supporting different languages) the product is said to be globalized. It is referred in short term as G11n.

## Localizability

It is an activity done after the globalization is complete. Also called as pseudo-localization, to test that you can easily translate the UI of the program to any target language without modifying code. Here you replace English characters with text containing non-English characters & if you achieve that successfully then that is an indication that your product is ready for localization. Finally you do localization testing to verify the quality of your product's localization for a particular target culture/locale. For example if one of your target language is Japanese, then you test Japanese version of your application in Japanese test environment itself. So in this phase you will test the localized version itself.

## The World-Readiness Initiative...

- 1 Its goal is to elevate products to the same level of internationalization, resulting in company-wide consistency.
- 2 The approach is divided into three phases, building on top of each other, spanning across three product cycles.
- 3 It is focusing on aspects of application "enablement" and "localizability".
- 4 The number of features per phase seems **achievable**.
- 5 Teams can handle features and requirements in a **flexible** way, depending on product schedule, focus, and markets

## Test planning

### Organizing test team

The World-Readiness testing effort must be spread throughout the team

- It is crucial that each Quality Engineers prepares the World-Readiness test cases for his/her assigned product areas, rather than having a single or a group of Quality Engineers dedicated to World-Readiness testing. The reason for this is because Globalization Quality Engineers cannot adequately know and test all areas of a complex product.
- Since World-readiness is an intrinsic quality of software (like stability, performance ease-of-use, etc.), all Quality Engineers must incorporate World-Readiness requirements into their routine and bug reporting practice.

### A team driver is required

- The effort needs to be presented and driven by a member of the Quality Engineers team, rather than by an external team (like the Globalization team).
- The designated Globalization Quality Engineers lead should be fully trained on the effort and will take on several responsibilities:
  1. Present the effort to teammates, using examples that are relevant to their product (rather than general concepts). This will help to ensure that the effort is perceived as relevant.

- 1 Ensure that the project is kept on rails, with every Quality Engineers on the team doing his part.
- 2 Train and assist all Quality Engineers on the basics.
- 3 Obtaining ad-hoc assistance from Globalization team.

**When does a team do World-Readiness testing?**

- It has to be defined in your overall Quality Plan
- Core functional Test Plan and Test Cases must include World-Readiness testing
- Start World-Readiness testing as soon as you start core functional testing

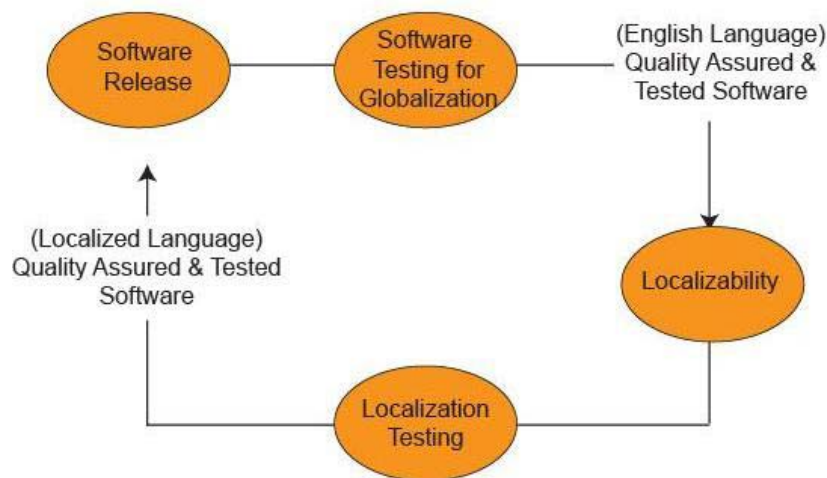
Remember to include testing on pseudo-translated build focusing on the UI parts to verify the localizability of the application under test.

Testing on pseudo-translated build can be performed during the allocated timeframe between Code Completed and UI Freeze.

The localization product developed would support the creation of pseudo-translated build for your product, you should work with them to incorporate creation of pseudo-translated build into your testing schedule.

**Test Cycle**

A typical test cycle to ship a world-ready product is shown below. As you can see, 1st you test your software for globalization. Here you make sure that your code can handle all international support without breaking functionality. You test your product that regardless of any international test environment, it behaves as per function specification.



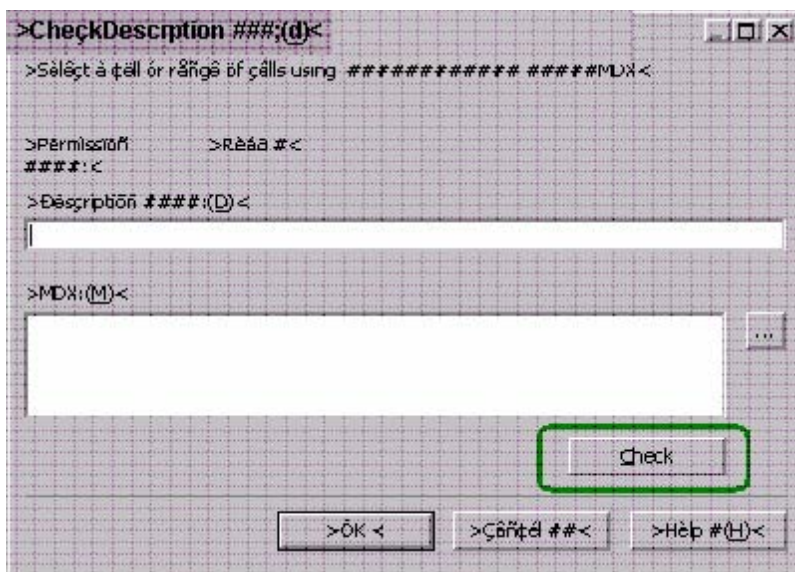
**Figure 3:- Test Cycle for making world ready application**

## Test Execution

### 1) Pseudo-localization or Mock build testing

It is also called as Localizability process where you can translate the UI of the program to any target language without re-engineering or modifying code. Your test team will use these Pseudo-localization version or mock builds to make sure that product is ready for localization. Pseudo-localization is perhaps the most cost-effective way of finding localizability bugs. Mock build testing is to ensure that 2 main UI issues are covered prior to localization .. Truncations .. Non Extracted Strings (hard coded strings)

The best approach to create the mock builds is where each string/message of your application is modified by adding a suffix & prefix. But the tip for your localizers is when you replace English text message with text containing non-English characters it is advisable to keep the text readable. This is required to overcome language limitations of your test team. For example if message on your original English application is “Save Filename as”. Then you can translate that as “>\*---Save Filename as --\*-<”.



**Figure 4: Example of Pseudo Localization or mock build**

### 2) Verifying with problematic characters

Below is the list of problematic characters which has high probability of finding errors. That means using these characters in your localization testing will increase chances of finding more bugs than using other characters of same language. For example, if you want to test support on Japanese version then you should use Japanese characters mentioned below for testing

Language	Few Problematic Characters (not all characters are listed)																		
French	æ (ALT+0230), Æ (ALT+0198), œ (ALT+0156), Œ (ALT+0140), à (ALT+0224), Á (ALT+0192), ä (ALT+0228), Ä (ALT+0196), ç (ALT+0231), Ç (ALT+0199), è (ALT+0232), È (ALT+0200), é (ALT+0233), É (ALT+0201), ì (ALT+0236), Ì (ALT+0204), í (ALT+0238), Î (ALT+0206), ò (ALT+0244), Ò (ALT+0212), ù (ALT+0249), Ù (ALT+0217), € (ALT+0128), « (ALT+0171), » (ALT+0187)																		
German	ß (ALT+0223), ü (ALT+0252), µ (ALT+0181), € (ALT+0128), ä (ALT+0228), Ä (ALT+0196), ö (ALT+0246), Ö (ALT+0214), € (ALT+0128)																		
Italian	à (ALT+0224), Á (ALT+0192), è (ALT+0232), È (ALT+0200), é (ALT+0233), É (ALT+0201), ì (ALT+0236), Î (ALT+0204), ò (ALT+0242), Ò (ALT+0210), ù (ALT+0249), Ù (ALT+0217)																		
Hindi	ोवन्ािुपगरक्तसलदजीीे्हनैव																		
Japanese	<table border="1"> <tr> <td>Boundary Cases</td> <td>濃濃黒</td> </tr> <tr> <td>Hankaku Katakana (Half-Width Katakana or SBCS)</td> <td>アカサタナハマヤラワン</td> </tr> <tr> <td>Zenkaku Katakana (Full-width Katakana or DBCS)</td> <td>アカサタナハ</td> </tr> <tr> <td>Zenkaku Hiragana (DBCS)</td> <td>あかさたなは</td> </tr> <tr> <td>Double-Byte Alphabet</td> <td>D o u b l e - B y t e</td> </tr> <tr> <td>Single-Byte Alphabet</td> <td>S i n g l e - B y t e</td> </tr> <tr> <td>Double-Byte Numbers</td> <td>1 2 3 4 5</td> </tr> <tr> <td>Single-Byte Numbers</td> <td>12345</td> </tr> <tr> <td>Kanji Characters (DBCS)</td> <td>日本語常用漢字</td> </tr> </table>	Boundary Cases	濃濃黒	Hankaku Katakana (Half-Width Katakana or SBCS)	アカサタナハマヤラワン	Zenkaku Katakana (Full-width Katakana or DBCS)	アカサタナハ	Zenkaku Hiragana (DBCS)	あかさたなは	Double-Byte Alphabet	D o u b l e - B y t e	Single-Byte Alphabet	S i n g l e - B y t e	Double-Byte Numbers	1 2 3 4 5	Single-Byte Numbers	12345	Kanji Characters (DBCS)	日本語常用漢字
Boundary Cases	濃濃黒																		
Hankaku Katakana (Half-Width Katakana or SBCS)	アカサタナハマヤラワン																		
Zenkaku Katakana (Full-width Katakana or DBCS)	アカサタナハ																		
Zenkaku Hiragana (DBCS)	あかさたなは																		
Double-Byte Alphabet	D o u b l e - B y t e																		
Single-Byte Alphabet	S i n g l e - B y t e																		
Double-Byte Numbers	1 2 3 4 5																		
Single-Byte Numbers	12345																		
Kanji Characters (DBCS)	日本語常用漢字																		
Spanish	Ñ (ALT+0241), ñ (ALT+0165)																		
Swedish	ä (ALT+0229), Ä (ALT+0197), å (ALT+0228), Å (ALT+0196), ö (ALT+0246), Ö (ALT+0214)																		

Figure 5: Examples of Problematic Characters

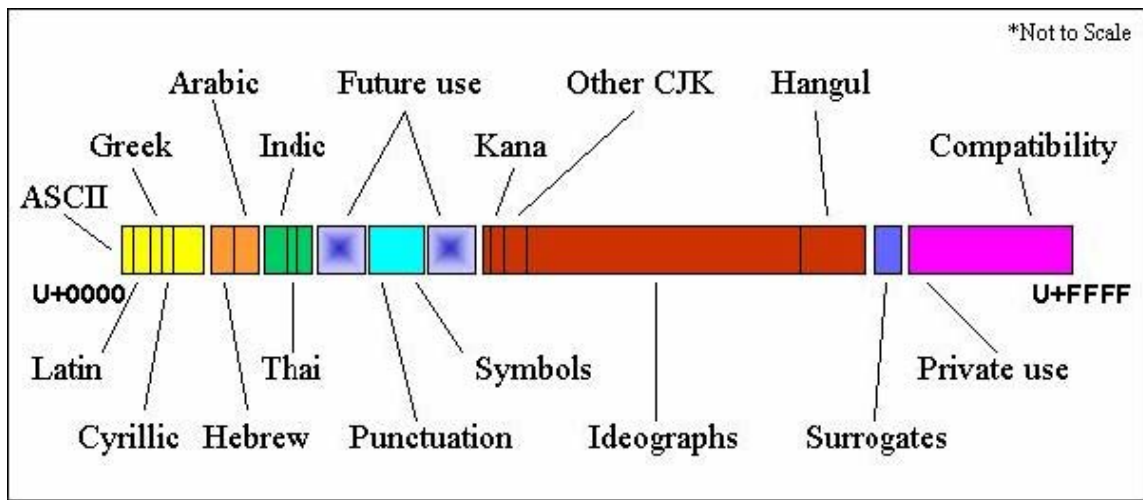


Figure 6: Multilingual User Interface (MUI)

### 3) Bug writing guidelines

Your Test or Quality plan should have clearly documented guidelines for writing localization bugs. Other than the standard bug writing guidelines there are few additional but mandatory fields which a tester should enter for writing localization

A good practice is to add 'Localization' keyword at the beginning of bug title in order to differentiate from general bug. Also one must add the 'locale OS', 'locale language' entry. By this not only it is easy for your team to differentiate localization bug from generic bug but it also helps to manage the count of localization bug found during the test cycle. Or it also helps to calculate how many new functional bugs are found during localization phase which were also open in original English version.

#### General localization guidelines

The following questions provide a good basis for testing an internationalized software product. non-ASCII characters are led from a resource file to the GUI without error.

- ☞ text buffers in the GUI are sufficiently sized for expansion
- ☞ if there are string length limitations, they are clearly indicated for the translators
- ☞ fonts for the GUI support all characters for the language being tested
- ☞ all text has been transferred to external resource files and is not "hardcoded"
- ☞ the resource file doesn't contain text **not** to be translated
- ☞ Keep an Eye Out for A "Locale-Neutral" GUI
- ☞ Watch for "Locale Neutral" Icons
- ☞ Watch for Symbols & Imagery
- ☞ Watch for National Flags
- ☞ Bi-Directional" (Right-to-Left) languages need careful GUI "mirroring" enablement tests

**Test Case Writing Tips (Localization Testing)** Be aware if your feature behaves differently in non-US versions. If any of the items below is applicable to your feature, please mention it in your test case.

#### Does your feature include?

- Online services (e.g., 3rd-party services) that are not available in non-US versions?
- Online services that are available but not in the target language?.
- Web links? (e.g. Help>Online Learning Resources) - The link should take you to a page in the target language or International English.
- Dates? - Date format is different depending on the locale.
- Name input fields? (e.g. Registration form, Contact Book) - In Japanese and Chinese family name comes before the first name.



- Address and/or phone number input fields? (e.g. Registration form, Contact Book)  
- Format varies depending on the country.
- Name sorting? (e.g. Albums/Tags, Contact Book) - Japanese character sorting is by pronunciation not by character code. So J version needs extra text field for pronunciation.
- Any symbols that may not make sense in non-US or western countries?
- Measurement units? (e.g. Preferences>Units & Rulers, Grid)
- Calendar? (e.g. Date View, Imperial calendar for J)
- Printing? - 1) Default paper size is A4 outside the US. 2) Localized versions may have different sets of picture package formats.
- Currency

## Sample bugs

### High severity/"Must-Test"

Install/uninstall not able to install/uninstall the pseudo-localized build  
Smoke Test Not able to complete a "smoke test" successfully without error

GUI Aesthetics Significant aesthetic issues in the majority of controls in a majority of dialogs 50% or more of all GUI controls exhibit expansion issues No "real estate" (room) to grow in a dialog with controls all tightly grouped Overlapped controls that block GUI on-screen text

GUI Hard-coding Strings still found in English after pseudo-localization ("hardcoded" strings) Non-translatable strings identified in resource files that, when pseudo-translated, break core functionality

### Medium severity/"Should-Test"

GUI Aesthetics Aesthetic issues in a minority of controls spread among a number of dialogs Approximately 25% of all GUI controls exhibit expansion issues Controls tightly grouped in a dialog with "real estate" (room) to grow Font in the GUI too small for the targeted language

Primarily an Asian character language issue

### Low severity/"Do-Not-Need-To Test"

GUI Aesthetics Aesthetic issues in a minority of controls within a minority of dialogs 10% or less of all GUI controls exhibit aesthetic issues Overlapped controls that do not block GUI on-screen text

## Summary

Today, every company is facing the challenges of a world that is connected and growing together via ubiquitous data access and exchange, 24/7 access to global resources is common. Accordingly, the profiles and, thus, the expectations of customers are changing. They are changing in that the customer not only require better, more flexible releases, but also an increased global functionality in a single product right "out-of-the-box". Not anymore is it enough to ship and sell one product for a single language. Instead, even if the user interface language is limited to a single or the preferred one, there is clearly the expectation that an application should be able to handle more than just language (mostly English or another language based on the Roman script): When copying, pasting, or importing from the web and other sources, it is likely that the imported sections also include some Cyrillic, Greek, or accented characters, maybe even a Japanese or Chinese character. The users expect at least the rendering of all of these to work, if not even the appropriate editing. Each popular product we know is expanding into new geographical markets and increase their products' language coverage. At the same time, they are shipping more languages in shorter sequence, even off-cycle. Microsoft, for example, is offering technologies like the "Language Interface Pack" (LIP) and the "Multilingual User Interface" (MUI) to provide optional language additions at any time.

## References

This paper has been carved as a result of our own work on various Adobe products in addition to taking one-line definitions of various terms from various sources.

## Biography



Abhishek Talwar works as Lead Software Engineer and has been with Adobe since last 5 years. He has a experience in various domains like Video, Imaging and Telecommunications. He has worked on various fields of testing including Test tool development, 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 and a patent publication. He has more than 10 international paper publications in various conferences around the world like SQS (UK), QA&Test(Spain), QAI(India), Test2008(India), PML (India). 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 Adobe QE quarterly newsletter.

Abhishek holds a Engineering 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.



Abhinav Agarwal (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.

Abhinav has written 5 international paper publications in various conferences around the world like SQS (UK), QA&Test(Spain) and Test2008(India).Abhinav holds a B Tech degree in Electronics Technology from Harcourt Butler Technological Institute, India .He is a qualified Adobe Certified Expert in one of Adobe's video Product.